

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**



PATENT ABSTRACTS OF JAPAN

(11) Publication number: **09274560 A**(43) Date of publication of application: **21.10.97**

(51) Int. Cl.

G06F 7/552**G06F 7/72****G06F 9/305****G09C 1/00**(21) Application number: **08110057**(22) Date of filing: **05.04.96**(71) Applicant: **OKI MICRO DESIGN MIYAZAKI:KK
OKI ELECTRIC IND CO
LTDKAWASAKI DENKI:GOUSHI**(72) Inventor: **EBIHARA HIDENORI
KAWASAKI KIYOTO**(54) **POWER REMAINDER OPERATION CIRCUIT,
POWER REMAINDER OPERATION SYSTEM AND
OPERATION METHOD FOR POWER REMAINDER
OPERATION**

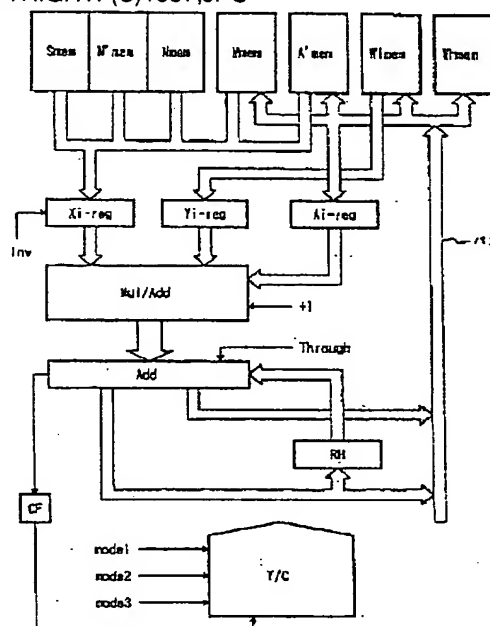
(57) Abstract:

PROBLEM TO BE SOLVED: To execute various kinds of operations only by supplying a mode signal prepared in advance in the case of finding the solution of power remainder operation.

SOLUTION: At the power remainder operation circuit of $M^e \bmod N$, this circuit is provided with an arithmetic part for executing the 1st operation of $A.A.R' \bmod N$ (R' is the inverse of R under the method of $\bmod N$) in response to a timing control signal corresponding to a 1st mode signal while using an integer R which is prime with N and larger than N , executing the 2nd operation of $A.B.R' \bmod N$ in response to a timing control signal corresponding to a 2nd mode signal while using the integer R and executing the 3rd operation of $A.1.R' \bmod N$ in response to a timing control signal corresponding to a 3rd mode signal while using the integer R and a timing control circuit T/C for outputting the timing control signal corresponding to

the 1st or 3rd mode signal to the arithmetic part.

COPYRIGHT: (C)1997,JPO



(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平9-274560

(43)公開日 平成9年(1997)10月21日

(51)Int.Cl. ⁸	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F	7/552		G 0 6 F	7/552
	7/72			7/72
	9/305	7259-5 J	G 0 9 C	1/00
G 0 9 C	1/00	6 5 0	G 0 6 F	9/30
				6 5 0 A
				3 4 0 A

審査請求 未請求 請求項の数12 F D (全 32 頁)

(21)出願番号 特願平8-110057

(22)出願日 平成8年(1996)4月5日

(71)出願人 591049893

株式会社神マイクロデザイン宮崎
宮崎県宮崎市大和町9番2号

(71)出願人 000000295

沖電気工業株式会社
東京都港区虎ノ門1丁目7番12号

(71)出願人 596060365

合資会社川▲崎▼電機
宮崎県児湯郡高鍋町大字南高鍋569番地3

(72)発明者 海老原 秀徳

宮崎県宮崎市大和町9番2号 株式会社神
マイクロデザイン宮崎内

(74)代理人 弁理士 佐藤 幸男 (外1名)

最終頁に続く

(54)【発明の名称】 べき乗剰余演算回路及びべき乗剰余演算システム及びべき乗剰余演算のための演算方法

(57)【要約】 (修正有)

【課題】 べき乗剰余演算の解を求めるに際し、予め用意されたモード信号を供給するのみで、各種演算を実行する。

【解決手段】 $M \bmod N$ の乗剰余演算回路において、 N と素であり、かつ N よりも大きい整数 R を用い、第1のモード信号に対応したタイミング制御信号に应答して、 $A \cdot A \cdot R' \bmod N$ (R' は $\bmod N$ 法下での R のインバース) なる第1の演算を実行し、整数 R を用い、第2のモード信号に対応したタイミング制御信号に应答して、 $A \cdot B \cdot R' \bmod N$ なる第2の演算を実行し、整数 R を用い、第3のモード信号に対応したタイミング制御信号に应答して、 $A \cdot 1 \cdot R' \bmod N$ なる第3の演算を実行する演算部と、第1ないし第3のモード信号に対応したタイミング制御信号を演算部に出力するタイミング制御回路 T/C とを設ける。

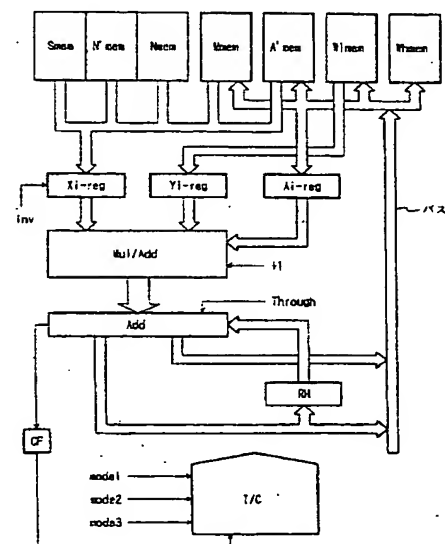


図1の実施の形態

【特許請求の範囲】

【請求項1】 正の整数 M 、 e 、 N に関する $M^e \bmod N$ なるべき乗剰余演算を行うべき乗剰余演算回路において、

N と素であり、かつ N よりも大きい整数 R を用い、第1のモード信号に対応したタイミング制御信号にตอบสนองして、 $A \cdot A \cdot R' \bmod N$ (R' は、 $\bmod N$ 法下での R のインバースである。)なる第1の演算を実行し、前記整数 R を用い、第2のモード信号に対応したタイミング制御信号にตอบสนองして、 $A \cdot B \cdot R' \bmod N$ なる第2の演算を実行し、前記整数 R を用い、第3のモード信号に対応したタイミング制御信号にตอบสนองして、 $A \cdot 1 \cdot R' \bmod N$ なる第3の演算を実行する演算部と、前記第1ないし第3のモード信号を受信し、前記第1ないし第3のモード信号に対応した前記タイミング制御信号を前記演算部に出力するタイミング制御回路とを有することを特徴とするべき乗剰余演算回路。

【請求項2】 前記第1ないし第3の演算の結果の前記 R 未満の値が0であることを検出する検出回路を設けたことを特徴とする請求項1記載のべき乗剰余演算回路。

【請求項3】 前記 A もしくは B のビット長に応じて、前記タイミング制御回路を制御する演算ビット長選択回路を設けたことを特徴とする請求項1記載のべき乗剰余演算回路。

【請求項4】 前記演算部は、第4のモード信号に対応したタイミング制御信号にตอบสนองして、 $A \cdot B + C$ なる第4の演算を実行し、前記タイミング制御回路は、前記第4のモード信号を受信し、前記第4のモード信号に対応した前記タイミング制御信号を出力することを特徴とする請求項1記載のべき乗剰余演算回路。

【請求項5】 前記演算における乗算値を格納する乗算値格納部と、前記演算における被乗算値を格納する複数の被乗算値格納部と、

複数の前記被乗算値格納部のいずれかの出力を選択して前記演算部に出力する選択回路と、

前記複数の被乗算値格納部に各々対応して設けられ、前記演算部の出力のうちの上位桁もしくは下位桁のいずれか一方を格納する複数の演算結果格納部とを設けたことを特徴とする請求項1記載のべき乗剰余演算回路。

【請求項6】 請求項1記載のべき乗剰余演算回路と外部装置との間に設けられ、前記外部装置の命令に応じて前記演算部の動作を制御する演算コントロール回路を設けたことを特徴とするべき乗剰余演算システム。

【請求項7】 前記第1、第2もしくは第3の演算のいずれかが終了したことを検出して、前記外部装置に割り込み要求信号を出力する割り込み制御回路を設けたことを特徴とする請求項6記載のべき乗剰余演算システム。

【請求項8】 前記第1、第2もしくは第3の演算のいずれかが実行中であることを検出して、スリープ信号を

出力するスリープ制御回路と、

前記スリープ信号を受信している間、クロック信号を前記外部装置に供給するクロック制御回路とを設けたことを特徴とする請求項6記載のべき乗剰余演算システム。

【請求項9】 第1のクロック信号を受信して、該第1のクロック信号を逡倍した第2のクロック信号を前記外部装置もしくは前記演算部に供給する逡倍速クロック制御回路を設けたことを特徴とする請求項6記載のべき乗剰余演算システム。

10 【請求項10】 正の整数 M 、 e 、 N に関する $M^e \bmod N$ なるべき乗剰余演算を、 N と素であり、かつ N よりも大きい整数 R を用いて実行するべき乗剰余演算の演算方法であって、乗算値を格納する乗算値格納部及び被乗算値を格納する被乗算値格納部とを有し、前記乗算値格納部に格納された乗算値と、前記被乗算値格納部に格納された被乗算値とを乗算した結果に対して $R' \bmod N$ (R' は、 $\bmod N$ 法下での R のインバースである。)を実行する演算回路を使用して、前記べき乗剰余演算のための演算方法において、

20 前記乗算値格納部及び前記被乗算値格納部に $R \bmod N$ を与えることにより第1の演算結果を得る第1の演算ステップと、

前記乗算値格納部に前記第1の演算結果を与え、前記被乗算値格納部に $R^2 \bmod N$ と M とを乗算した演算結果を与えることにより第2の演算結果を得る第2の演算ステップと、

30 前記乗算値格納部に前記第1の演算結果もしくは前記第2の演算結果を与え、前記被乗算値格納部に1を与えることにより第3の演算結果を得る第3の演算ステップとを有し、

前記第1ないし第3の各演算ステップにおける演算は、前記第1ないし第3の各演算結果を X として、

$m = (X \bmod R) \cdot N' \bmod R$ なる第4の演算 (N' は、 $\bmod N$ 法下での N のインバースである。)を実行する第4の演算ステップと、その後、

$t = (X + m \cdot N) / R$ なる第5の演算を実行する第5の演算ステップとを含むことを特徴とするべき乗剰余演算のための演算方法。

40 【請求項11】 前記第4の演算は、 X の R 未満の値と N' とを乗算し、この乗算結果の R 未満の値を m とするステップからなり、

前記第5の演算は、 X の R 未満の値が全て0であることを検出した時には、この X の R 以上の値を t とし、 X の R 未満の値が少なくとも1を含むことを検出した時には、 $(X \text{の} R \text{以上の値} + m \cdot N \text{の} R \text{以上の値}) + 1$ を t とするステップからなることを特徴とする請求項10記載の演算方法。

【請求項12】 前記第5の演算ステップはさらに、 $R \leq t$ の関係を検出した時は $t - N$ 実行することを特徴とする請求項10記載の演算方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、情報通信ネットワーク、交通、金融、医療、流通等の分野において使用される情報の暗号化技術及び復号化技術に関するものであり、特にこれらの情報の暗号化及び復号化を実現するためのべき乗剰余演算回路及びそのシステム及びべき乗剰余演算のための演算方法に関するものである。

【0002】

【従来の技術】情報通信技術の発展に伴い、情報ネットワーク上のセキュリティーの確保（データの盗用や破壊を防止する）が重要視されるようになってきている。そのために、情報の暗号化技術及び復号化技術が、情報通信分野にとどまらず、交通、金融、医療、流通等の身近な分野で使用されつつある。従って、この種の暗号化技術及び復号化技術は、高度なセキュリティーが単純な原理によって実現できることが要求される。まず、この種の技術の理解を容易にするため、情報の暗号化・復号化についての概略を説明する。

【0003】暗号の世界においては、“非対象アルゴリズム”が質的に優れている。非対象暗号アルゴリズムとは、暗号化鍵と復号化鍵が異なり、そのいずれか一方から他方が容易に計算できない暗号アルゴリズムをいう。

【0004】この非対象暗号アルゴリズムの代表的なものに、べき乗剰余演算を用いるタイプのRSA暗号、エルガマル暗号、ラビン暗号、ウィリアムス暗号等がある。そして、暗号アルゴリズムを応用する上では、“デジタル署名”のシステムがあり、現在その標準化の動きがある。その対象となっている代表的なものは、RSA署名法、エルガマル署名法、シュノア署名法、DSA (Digital Signature Algorithm) 署名法等であり、これらは全て長いビット長のべき乗剰余演算を使用するタ

$$n = p \cdot q \cdots (4)$$

$$1 \equiv e \cdot d \pmod{\text{LCM}(p-1, q-1)} \cdots (5)$$

(≡は、左辺と右辺が相似であること、LCMは、最小公倍数を意味する。またpとqとは互いに素な整数である。)なる条件があらかじめ与えられている。なお、e、nは公開鍵、d、p、qは秘密鍵である。

【0011】以上の式(4)(5)は、ともに暗号アルゴリズム上のべき乗剰余演算の数値の条件を定義しており、式(4)は、nは互いに素な大きな素数pとqとの積であることを示している。今、pとqはともに奇数なので、当然nは奇数でなければならない。次に式(5)は、式(4)で示したp及びqの1だけ小さい値同士の最小公倍数でeとdとの積e・dを割ったときの余りが1になることを示している。

【0012】以上の式(4)(5)のような条件に基づき、平文Mは式(2)を用いて暗号化され、また暗号化された平文M(暗号文C)は式(3)を用いて復号化さ

*イプのものである。従って、これらのデジタル署名のシステムを実現する上では、長いビット長のべき乗剰余演算を短時間で終了することのできる演算器の開発が必要不可欠である。

【0005】上で説明したRSA暗号、エルガマル暗号、ラビン暗号、ウィリアムス暗号等は、次式(1)のべき乗剰余演算の形式が基本として使用される。式

(1)は、 X^Y をNで割った時の余りを求めることを意味する。また、式(1)において、Xは暗号化(復号化)の対象となる平文、Y及びNは暗号化(復号化)のための鍵(キー)である。

$$[0006] X^Y \pmod N \cdots (1)$$

このべき乗剰余演算を用いることにより、情報の暗号化及び復号化が容易に実行され、かつX、Y、Nのオペランドビット長を長くすることで、各鍵の解読を困難にすることができる。

【0007】しかし、オペランドビット長を長くすると、べき乗剰余演算に長時間を要することになる。そこで、オペランドビット長が長いべき乗剰余演算をいかに短時間に終了させるかがポイントとなる。

【0008】さて、RSA暗号を例にとり、べき乗剰余演算を使用した実際の暗号化・復号化、その使われ方を以下に説明する。

【0009】(1) RSA暗号の暗号化・復号化の概略暗号化には、

$$C = M^e \pmod n \cdots (2)$$

なる式を使用する。

【0010】復号化には、

$$M = C^d \pmod n \cdots (3)$$

なる式を使用する。ここでMは、暗号化の対象となる平文、Cは暗号化された平文すなわち暗号文である。そして、式(2)におけるe、nは暗号化鍵、式(3)におけるd、nは復号化鍵であり、

れる。

【0013】(2) 暗号化・復号化の事例

次に具体的な事例として、「発信者Aは平文Mを暗号文Cに暗号化して送信し、受信者Bは暗号文Cを平文Mに復号化する。」という伝達(デジタル署名あり)を行った場合の、発信者A及び受信者Bが行う処理方法を図2を用いて説明する。

【0014】送信者Aが行う処理

・自分が作成した平文MAを自分の秘密鍵 d^A を使って変形し、署名文CAを作成する。(署名)

$$CA \equiv MA^{d^A} \pmod n \cdots (6)$$

・Bの公開鍵 e^B を使って、暗号化署名文cAを作成する。(暗号化)

$$cA \equiv CA^{e^B} \pmod n \cdots (7)$$

・cAをBへ送信する。

【0015】受信者Bが行う処理

*て変形する。(復号化)

・受信した暗号化署名文cAを自分の秘密鍵dBを使つ*

$$cA^{dB \bmod nB} \equiv (CA^{eB \bmod nB})^{dB \bmod nB} \cdots (8)$$

ここで、 $CA^{eB} = X$ とおくと、式(8)は、

$$(CA^{eB \bmod nB})^{dB \bmod nB} = (X \bmod nB)^{dB \bmod nB} \cdots (9)$$

と変形できる。

$$\ast X = k \cdot nB + Y$$

【0016】ここで、式(9)中の $X \bmod nB = Y$ と

$$Y = X - k \cdot nB \cdots (10)$$

おく、すなわちXをnBで割ったときの余りがYであり
そのときの商がkとすると、

と変形できる。よって、式(10)を式(9)の右辺に

※10 代入すると、

$$\begin{aligned} (X \bmod nB)^{dB \bmod nB} &= Y^{dB \bmod nB} \\ &= (X - k \cdot nB)^{dB \bmod nB} \cdots (11) \end{aligned}$$

となる。式(11)の $(X - k \cdot nB)^{dB}$ を展開する。★ ★と、定数 a_i ($i=1, 2, \dots$)を用いて、

$$\begin{aligned} (X - k \cdot nB)^{dB} &= (X^{dB} - a_1 \cdot X^{dB-1} \cdot nB + a_2 \cdot X^{dB-2} \cdot nB^2 - \\ &\quad \cdots - a_i \cdot nB^{dB}) \cdots (12) \end{aligned}$$

と表すことができる。この式(12)を式(11)に代☆ ☆入すると、

$$\begin{aligned} (X \bmod nB)^{dB \bmod nB} &= Y^{dB \bmod nB} \\ &= (X - k \cdot nB)^{dB \bmod nB} \\ &= (X^{dB} - a_1 \cdot X^{dB-1} \cdot nB + a_2 \cdot X^{dB-2} \cdot nB^2 - \\ &\quad \cdots - a_i \cdot nB^{dB}) \bmod nB \\ &= X^{dB \bmod nB} - a_1 \cdot X^{dB-1} \cdot nB \bmod nB \\ &\quad + a_2 \cdot X^{dB-2} \cdot nB^2 \bmod nB - \cdots \\ &\quad - a_i \cdot nB^{dB \bmod nB} \end{aligned}$$

この式中の第2項以降は、全てnBで割り切れるので、削除できる。よって、

$$= X^{dB \bmod nB} \cdots (13)$$

となる。先に $CA^{eB} = X$ としたので、元に戻すと、

$$= (CA^{eB})^{dB \bmod nB} \cdots (14)$$

が得られる。

◆足するので、ある整数hを用いて

【0017】ここまでの過程をまとめると、

$$30 \quad eB \cdot dB = h(pB - 1) + 1$$

$$\begin{aligned} cA^{dB \bmod nB} &\equiv (CA^{eB \bmod nB})^{dB \bmod nB} \\ &= (CA^{eB})^{dB \bmod nB} \end{aligned}$$

と表せる。ここで、素数pと、pと互いに素な任意の整数Xに対して

となる。

$$XP^{-1} \bmod p = 1$$

【0018】さて、上記のeB、dBは、式(5)を満た◆

が成立するというフェルマの小定理を用いると、

$$\begin{aligned} CA^{eB \cdot dB \bmod pB} &= CA^{h(p-1)+1 \bmod pB} \\ &= CA \cdot CA^{h(p-1) \bmod pB} \\ &= CA \bmod pB \cdots (15) \end{aligned}$$

となる。CAがpBの倍数でも上式を満足するので、全てのCAについて $CA^{eB \cdot dB} - CA$ は、pBで割り切れる。同様に $CA^{eB \cdot dB} - CA$ はqBでも割り切れる。pBとqBは、異なる素数なので、 $CA^{eB \cdot dB} - CA$ は $nB = pB \cdot qB$ でも割り切れる。よって、 $cA^{dB \bmod nB} \equiv CA^{eB \cdot dB \bmod nB} \equiv CA \bmod nB (=CA)$

が成立する。

【0019】送信者の公開鍵eAを使用して、平文MAを作成する。(署名認証)

$$\begin{aligned} CA^{** \bmod nA} &\equiv (MA^{**})^{** \bmod nA} \\ &\equiv (MA^{**})^{** \bmod nA} \end{aligned}$$

上記した復号化処理と同様にして計算すると、

が導き出される。

【0020】以上のように、e、d、nの値を式(4)(5)のような条件下で決定し、式(1)のようなべき乗剰余演算の形式を基本として使用することにより、平文を暗号化した暗号化された平文を復号化することができる。

【0021】例えば、 $n=15$ 、 $e=3$ 、 $p=5$ 、 $q=3$ 、 $d=11$ とし、

7

8

$$(n=p \cdot q=5 \cdot 3=15,$$

$$e \cdot d \bmod (p-1) \cdot (q-1) = 3 \cdot 11 \bmod 4 \cdot 2 \\ = 33 \bmod 8 \\ = 1)$$

平文 $M=13$ とすると、

$$\text{暗号化: } C = M^e \bmod n = 13^3 \bmod 15 = 2197 \\ \bmod 15 = 7$$

$$\text{復号化: } M = C^d \bmod n = 7^{11} \bmod 15 = 1977 \\ 326743 \bmod 15 = 13$$

となり、平文 $M=13$ が復号化されたことが確認された。

【0022】(3) べき乗剰余演算の演算方法
次に、暗号化・復号化で使用する、べき乗剰余演算の演算方法を説明する。

【0023】 $A = M^e \bmod N$ のべき乗剰余演算は、整数 e の2進数展開を $e = e_{k-1} \cdot \dots \cdot e_1 e_0$ として、以下のフロー1に示す反復平方積法を使用して実行される。

【0024】

```

[フロー1]
begin
  A=1
  for i=k-1 down to 0 do
    begin
      A=A2 mod N ... (16)
      if ei=1 then A=A·M mod N ... (17)
    end
  end
end

```

そして、この反復平方積法をフローチャートで表現すると、図3のようになる。

【0025】まず、初期値1をAレジスタに格納する。次に、Aレジスタに格納された値を乗算して $A \times A$ を求め、この $A \times A$ をNで除算して余りを求め、この値をレジスタaに格納する。次に、レジスタaに格納された値をレジスタAに格納する。このとき、もし指数 e が1ならば、レジスタAに格納された値と平文Mを乗算した後Nで除算して余りを求め、この値をレジスタaに格納する。そして、このレジスタaの内容を再度レジスタAに格納する。もし、指数 e が0ならば、以上の計算は行わずに、レジスタAに格納された値には何も施さない。この演算を e の最上位ビットから最下位ビットまで繰り返し行い、最終的にレジスタAに格納された値が、求めたいべき乗剰余演算の解になる。

【0026】以上のように、演算の基本は、式(16)

(17)に示すように乗算と除算(mod算)である。乗算は、初期値を1とするAの値に対して $A \times A$ あるいは $A \times M$ を行う部分であり、除算は、各々の乗算で得られた値に対して $\bmod N$ を行う部分である。この”乗算と除算”($A \times A \bmod N$ 、 $A \times M \bmod N$)を一對の演算として、” e ”のビット値に従って繰り返す。つま

り、” e ”の最上位ビットから最下位ビットまでの各ビットの内容によって”乗算と除算”を行うのである。

【0027】

【発明が解決しようとする課題】べき乗剰余演算は、基本となる剰余演算(mod算)を繰り返し行うことで解を得ることができることを示したが、この繰り返し回数自体は、ただだか数百〜数千回であるので、ソフト的な処理でも充分に対応できる。しかしながら、この剰余演算自体すなわち除算を実行するためには、大規模な演算回路と複雑な処理手順が必要とされるため改善が望まれていた。

【0028】

【課題を解決するための手段】本発明の代表的なものは、正の整数 M 、 e 、 N に関する $M^e \bmod N$ なるべき乗剰余演算を行うべき乗剰余演算回路において、 N と素であり、かつ N よりも大きい整数 R を用い、第1のモード信号に対応したタイミング制御信号に応答して、 $A \cdot A \cdot R' \bmod N$ (R' は $\bmod N$ 法下での R のインバースである。)なる第1の演算を実行し、前記整数 R を用い、第2のモード信号に対応したタイミング制御信号に応答して、 $A \cdot B \cdot R' \bmod N$ なる第2の演算を実行し、前記整数 R を用い、第3のモード信号に対応したタイミング制御信号に応答して、 $A \cdot 1 \cdot R' \bmod N$ なる第3の演算を実行する演算部と、前記第1ないし第3のモード信号を受信し、前記第1ないし第3のモード信号に対応した前記タイミング制御信号を前記演算部に出力するタイミング制御回路とを設けたものである。

【0029】

【発明の実施の形態】べき乗剰余演算は、基本となる剰余演算(mod算)を実行する手順が非常に複雑であるため、演算回路が大規模化してしまうことを前述した。そこで、Montgomery (モンゴメリー) は、剰余演算(mod算)を先のような一般的な方法で行わずに、”乗算”と簡単なビット列処理を行うことによって、剰余演算の解を得る仕組みを提案している。本発明は、基本的にこのモンゴメリーが提案している仕組みを利用して演算を行うものであるため、以下にモンゴメリーが提案している手法について簡単に説明するが、各演算において、演算時間を短縮するための工夫は、本願特有のものであることを述べておく。

【0030】剰余演算において、 R をモジュラス N より少しだけ大きい2の指数と定義し、”乗算 $\bmod N$ ”

〔ある値 \times ある値〕 \div (値 N)の余りを求めるという意味。〕の法下での R のインバース値を R' と定義し、($R \cdot R' \bmod N = 1$ が成立する。)さらに、 $R \cdot R' - N \cdot N' = 1$ 、 $0 < N' < R$ なる関係を満足する

N' (N' は、"乗算 mod N " の法下での N のインバース値である。) を定義する。この時に、例えば、 $M(X) = X \bmod N$ 形式の剰余演算を行う場合に、 $M'(X) = X \cdot R' \bmod N \cdots (18)$ の形に置き換えて、次のフロー 2 に示す REDC (X) 関数の計算方法を実行すれば、剰余演算 (mod 算) の解を先の一般的方法 (乗算と除算とを単に実行するもの) によらずに得ることができるというものである。ただし、フロー 2 は剰余演算の解を求めるフローであり、べき乗剰余演算の解を求めるフローではない。なお、上記関数で得られた t か $t - N$ が式 (18) の解である。
【0031】

(フロー 2)

REDC (X) 関数

function REDC (X)

$m = (X \bmod R) \cdot N' \bmod R \cdots (19)$

$t = (X + m \cdot N) / R \cdots (20)$

if $t < N$

return $t \cdots (21)$

else return $t - N \cdots (22)$

20

以上の関数を見ると、演算要素が N 及び N' を使用した *

$$\begin{aligned} m \cdot N &= ((X \bmod R) \cdot N' \bmod R) \cdot N \\ &\equiv X \cdot N \cdot N' \bmod R \\ &\equiv X \cdot (R \cdot R' - 1) \bmod R \\ &\equiv X \cdot R \cdot R' \bmod R - X \bmod R \\ &\equiv -X \bmod R \\ X + m \cdot N &\equiv X + (-X \bmod R) \\ &\equiv 0 \bmod R \end{aligned}$$

となる。この式は、 $X + m \cdot N$ を R で割ったときの余りが 0、言い替えると、 $X + m \cdot N$ は、 R で割り切れることを意味する。

【0033】ここで、 $X + m \cdot N$ は、" X " と " N の乗算 " との和であるから、

$$(X + m \cdot N) \bmod N \equiv X \bmod N + m \cdot N \bmod N \equiv X \bmod N$$

よって、式 (20) より、

$$t \cdot R \bmod N \equiv X \bmod N$$

この両辺を R' 倍すると、

$$t \cdot R \cdot R' \bmod N \equiv t \bmod N \equiv X \cdot R' \bmod N$$

【0034】ところで、上記式 $X \cdot R' \bmod N$ 中の X が、 $\bmod N$ あるいは $\bmod R$ を計算した後の乗算値であるならば、(X が、 $\bmod N$ あるいは $\bmod R$ を計算した後の乗算値であることは後に説明する。)

$$X < N \cdot N < R \cdot N < R \cdot R$$

m は $\bmod R$ を実行した結果であるから、 $m < R$ である。よって、 $m \cdot N < R \cdot N$ となる。

【0035】 $X < R \cdot N$ であるから、 $X + m \cdot N < R \cdot N + R \cdot N = 2R \cdot N$ である。従って、

$$t = (X + m \cdot N) / R < 2N$$

* 乗算と、 R を用いた除算で構成されている。 $R = 2^n$ という定義なので、 R を用いた除算は、被除算値の 2^n 以上の値が商、 2^n 未満の値が余りを示す。従って、式 (19) における \bmod 算においては、基本的に 2^n 未満の値を見るだけでよく、式 (20) における除算においては、基本的に 2^n 以上の値を見るだけでよい。つまり、実質的に除算 (剰余演算) を行うことなく乗算と加算のみで剰余演算の解を求めることができる。

【0032】次に参考のために、 m 、 R 、 N との間の関係を式 (19) (20) を使用して説明する。式 (19) において、

$$t < 2N$$

が成立する。

【0036】そして、もし t が N より大きい値の場合、両辺から N を引くと、

$$t - N < N$$

となり、この式から $t - N$ は、 $\bmod N$ を施した値となることがわかる。

【0037】さらに、式 (20) の $t = (X + m \cdot N) / R$ の計算と、 t の計算終了について補足する。

【0038】 $X + m \cdot N$ は、必ず R の倍数であるから、例えば、 $R = 2^{576}$ とすると、 $X + m \cdot N$ の 576 ビット未満の値は全て 0 である。従って、式 (20) の t の計算は次の 2 通りに分けられる。

【0039】・ X が R の倍数でない場合

$X + m \cdot N$ と R のビット列イメージを示すと以下のようになる。

$$[0040] R = 1000 \cdots 0000$$

$$X = \text{????????} \cdots \text{????}$$

$$m \cdot N = \text{????????} \cdots \text{????}$$

$$X + m \cdot N = \text{??????}000 \cdots 0000$$

式 (20) の計算において、 t は必ず整数値が得られるので、 $X + m \cdot N$ は R の倍数になる。従って、 $X + m \cdot$

50

Nは、上記ビット列イメージのようになる。そして、アンダーラインを付与した部分がtの解となる。その理由は、 $X+m \cdot N$ はRの倍数であるから、X及び $m \cdot N$ のR値未満の加算結果は必ず0になるはずで、しかもそれらの?で示す値は元々オール0ではないので、加算の課程でR値以上の桁への桁上げが生じているはずである。よって、この場合、X及び $m \cdot N$ 値のR値未満の値を無視して、

$$t = (X \text{ 値の } R \text{ 以上の値}) + (m \cdot N \text{ 値の } R \text{ 以上の値}) + 1$$

の計算で解が得られる。

【0041】・XがRの倍数である場合

XとRのビット列イメージを示すと以下ようになる。

【0042】 $R=1000 \cdots 0000$

$X=?????000 \cdots 0000$

このとき、式(19)中の $X \bmod R$ は0となるので、 $m=0$ となる。 $m=0$ ならば式(20)の計算は、 $t=X/R$ で良いことになる。ここで、tは必ず整数値が得られるので、

$$t = (X \text{ 値の } R \text{ 以上の値})$$

の計算で解が得られる。

【0043】X値がRの倍数であるか否かの判別は、X値のR未満の値が0か否かで判別できる。このことは、tの計算においてX値の全ビット長分を計算する必要がなく、計算量の縮小と計算時間の短縮が図れることを意味する。

【0044】tの計算終了に関しては、REDC(X)関数の演算結果tは $0 < t < 2N$ の範囲内の値となり、 $N < t$ の時再度 $t-N$ の計算を行わなければならないことをフロー2で示した。しかし、べき乗剰余演算の途中で行われたREDC(X)関数の演算結果tについては、 $N < t$ の関係にあっても $t < R$ の範囲であれば、そのまま後の演算に進んでも構わない。なぜなら、後のmod算でここで取り残されたN値を除去するからである。

【0045】いま、べき乗剰余演算の途中の演算結果を*

$$\begin{aligned} A \cdot A \cdot R' \bmod N &= (N+2) \cdot (N+2) \cdot R' \bmod N \\ &= (N^2+4N+4) \cdot R' \bmod N \\ &\equiv N^2 \cdot R' \bmod N + 4N \cdot R' \bmod N + 4 \cdot R' \bmod N \\ &\equiv 4 \cdot R' \bmod N \end{aligned}$$

となる。これは仮に前回のREDC(X)関数で得たtの値が $t=N+2$ ではなくmod N算の純粋な解 $t=2$ で $A \cdot A \cdot R' \bmod N$ の計算を行なった場合と同じ結果となる。

【0050】(実際例) $X=44123$, $R=28$, $N=199$ の時の $M'(X)=X \cdot R' \bmod N$ の計算。

【0051】 $R \cdot R' \bmod N=1$, $R \cdot R' - N \cdot N'=1$ の関係から $R'=7$, $N'=9$ 。従って、 $M'(X)=44123 \times 7 \bmod 199=13=0D_H$ となる筈である。ここでREDC(X)関数を使用して上

* $M \bmod N=S$ とすると、

$$M=k1 \cdot N+S=(k1-1) \cdot N+N+S$$

となる関係が成立しているから、今回のmod算でN+Sを取り残したと考えると次のmod算、例えばべき乗剰余演算の $A^2 \bmod N$ (図3参照)にて

$$\begin{aligned} M1 &= A \cdot A = (N+S) \cdot (N+S) \\ &= N^2+2 \cdot N \cdot S+S^2 \end{aligned}$$

という処理を施して、再度mod算を行う。

【0046】

$$\begin{aligned} 10 \quad M1 \bmod N &= (N^2+2 \cdot N \cdot S+S^2) \bmod N \\ &\equiv S^2 \bmod N \end{aligned}$$

となり、これは前のmod算で取り残したN値を除去していることを意味する。また、M1をべき乗剰余演算の $A \cdot M \bmod N$ にて計算しても同様である。モンゴメリ手法下でも同様のことが言える。

【0047】簡単のために低ビット長の実際の数値例で説明する。

【0048】 $N=13(1101)$, $R=(10000)$, 前回のREDC(X)関数で得られた値 $t=15(1111)$ とすると、これらは、 $N \leq t < R < 2N$ の関係にある。この時のtのbit4は"0"であるが、tの値自体未だmod算の純粋な解は得られていない。しかし、tはNのビット長を越えていないので、次のべき乗剰余演算の $A \cdot A \cdot R' \bmod N$ あるいは $A \cdot M \cdot R' \bmod N$ のべき乗剰余演算の上でAに代入する値としては妥当である。つまり、次に行なわれる乗算 $A \cdot A$, $A \cdot M$ の結果の値が規定のビット長(上記の数値例では4ビット $\times 2=8$ ビット)を越えないので、ビット長が規定された"乗算 $R' \bmod N$ "の演算を継続して行なうことができる。

【0049】ところで、 $t=15=13+2=N+2$ と表せる。このことはmod Nの計算の上では前回のREDC(X)関数で得たtの値は"Nを取り残した値"であると言える。このtの値のまま、Aに代入して $A \cdot A \cdot R' \bmod N$ の計算を行なうと、

記解を得る。

【0052】 $X=44123=AC5B_H$, $R=100_H$, $N=199=C7_H$, $N'=9_H$ より、(REDC(X)関数)

$$m = (X \bmod R) \cdot N' \bmod R = 5B_H \times 9_H \bmod R = 333_H \bmod R = 33_H$$

$$t = (X+m \cdot N) / R = (AC5B_H + 33_H \times C7_H) / R = D400_H / R = D4_H$$

純粋な解は、 $t-N=D4_H-C7_H=0D_H$ であるがこのまま $t \cdot t \cdot R' \bmod N$ を実行してみる(次式1)

13

1、 $t = D4_H \rightarrow t \cdot t = D4_H \times D4_H = AF90_H$ の場合

$m = (X \bmod R) \cdot N' \bmod R = 90_H \times 9_H \bmod R = 510_H \bmod R = 10_H$

$t = (X + m \cdot N) / R = (AF90_H + 10_H \times C7_H) / R = BC00_H / R = BC_H$

2、 $t = 0D_H \rightarrow t \cdot t = 0D_H \times 0D_H = 00A9_H$ の場合

$m = (X \bmod R) \cdot N' \bmod R = A9_H \times 9_H \bmod R = 5F1_H \bmod R = F1_H$

$t = (X + m \cdot N) / R = (00A9_H + F1_H \times C7_H) / R = BC00_H / R = BC_H$

上式1、2の計算結果は、アンダーラインのように同値となる。

【0053】このことは、演算時間短縮を図る上では重要である。つまり、Nの最上位ビットを $\text{bit}(n-1)$ とするとき、得られたtの $\text{bit}(n)$ が1の時 $t-N$ を行い、0の時は後の演算に進むようにすれば良い。

【0054】一般に576ビット長オペランドのべき乗剰余演算の場合は、 $R = 2^{576}$ にする。そのとき、式

(20)の計算結果が576ビット長を越えた場合(576ビットの桁あふれが生じた場合)が、 $t \geq R$ を示す。また、その場合の $t-N$ の計算は、

$t-N = (t \text{ 値の下位576ビットの値}) + (N \text{ のインバース値}) + 1$

で求める。暗号アルゴリズムの世界でのNの値は奇数であるから、Nのインバース値+1は、Nのインバースを求めてその最下位ビットを1にする操作で得られる。

【0055】さて、フロー2で示した、モンゴメリー演*

[フロー3]

begin

$A = 1 \cdot R \bmod N = R \bmod N \quad \dots (23)$

for $i = k-1$ down to 0 do

begin

$A = A^2 \cdot R' \bmod N \quad \dots (24)$

if $e_i = 1$ then $A = A \cdot (M \cdot R \bmod N) \cdot R' \bmod N \quad \dots (25)$

end

$A = A \cdot R' \bmod N \quad \dots (26)$

end

フロー1とフロー3との大きな違いは、上記式(23)において、レジスタAに格納する初期値を1とせず、後のモンゴメリー演算を考慮して $R \bmod N$ ($1 \cdot R \bmod N$)を格納している点、上記式(25)において、前記式(17)のMの代わりに $M \cdot R \bmod N$ の値を用いている点及び上記式(26)を新たに行う点である。レジスタAにあらかじめ $R \bmod N$ を格納しておくことにより、式(24)(25)(26)における R' が除去されるのである。そして、フロー3の反復平方積法をフローチャートで表現すると、図1ようになる。

【0061】演算の実行に先だて、この図1において

14

*算を実行するためのREDC関数は、モンゴメリー演算の解を得ているにすぎない。すなわち、式(18)に示すように、計算を容易にするため R' という特有の数値を使用している。そもそも求めたいのは、 R' を含まない乗算 $\bmod N$ 形式を使用した式の解であるので、フロー2で示すモンゴメリー演算の解を、何がしかの操作で R' を含まない数値に戻す必要がある。

【0056】そこで、本願では、次のような性質を利用して、モンゴメリー演算特有の上記 R' という値を相殺した。

【0057】今、 $M'(X) = X \cdot R' \bmod N$ において、Xが $(X \cdot R)$ であるとき、

$M'(X \cdot R) = (X \cdot R) R' \bmod N = X \bmod N$

Xが $(X \cdot R \bmod N)$ であるとき、

$M'(X \cdot R \bmod N) = (X \cdot R \bmod N) R' \bmod N = X \bmod N$

となる。

【0058】つまり、 $? \cdot R$ あるいは $? \cdot R \bmod N$ の形のものにモンゴメリー演算を適用すると、モンゴメリー演算特有の R' が除去されて、 $? \bmod N$ の形になる。

【0059】そして、この性質をフロー1で示した、反復平方積法を使用した、べき乗剰余演算に応用すれば、フロー3で示すように、べき乗剰余演算の解が得られる。正確には、フロー3で示す式(24)(25)(26)の $\bmod N$ 算の各々において、フロー2を実行することにより、各式(24)(25)(26)における解が容易に得られ、結果としてフロー3を使用した、べき乗剰余演算の解が容易に得られる。

【0060】

あらかじめ必要とされる $R \bmod N$ 、 $M \cdot R \bmod N$ 、 N' を求めておかなければならない。

【0062】 $[R \bmod N \text{ について}]$ RSA暗号等に用いられるべき乗剰余演算においては、 N の最上位ビット b_{n-1} と最下位ビット b_0 は1である。従って、そのときのRは、 2^n に選ばれる。よって、 $R \bmod N = R - N$ となる。この $R - N$ は、Nのインバースを求めて、その最下位ビットを1にすることで容易に求められる。

【0063】 $[N' \text{ について}]$ $R \cdot R' \bmod N = 1$ 、 $R \cdot R' - N \cdot N' = 1$ 、 $0 < R'$ 、 $N' < R$ の関係より、

$R' < N'$ 、 $N' - R < R - N$

なる関係を導くことができる。

【0064】暗号システムの運用者は、これらの関係を満足するように、 N 値に対応する N' 値を決定する。す*

$$M \cdot (R^2 \bmod N) \cdot R' \bmod N \equiv M \cdot R^2 \cdot R' \bmod N \\ \equiv M \cdot R \bmod N$$

により、 $M \cdot R \bmod N$ を求めることができる。

【0066】例えば、今、 $R = 2^{512}$ とすると、
 $2^{513} \bmod N = 2 \cdot 2^{512} \bmod N \equiv 2 \cdot (R \bmod N) \bmod N = A$

とすると、この A を初期値として以下のフロー4を実行すると $R^2 \bmod N$ の解を求めることができる。

【0067】

【フロー4】

```
begin
  for i=1 to 9
    begin M(A,A)=A·A·R' mod N
    end
  end
```

このフロー4中の計算課程は、以下の通りである。 ※20

i=1 $A = 2^{513} \bmod N$ $M(A,A) = 2^{513} \cdot 2^{513} \cdot 2^{-512} \bmod N \equiv 2^{514} \bmod N$
 i=2 $A = 2^{514} \bmod N$ $M(A,A) = 2^{514} \cdot 2^{514} \cdot 2^{-512} \bmod N \equiv 2^{516} \bmod N$
 i=3 $A = 2^{516} \bmod N$ $M(A,A) = 2^{516} \cdot 2^{516} \cdot 2^{-512} \bmod N \equiv 2^{520} \bmod N$
 i=4 $A = 2^{520} \bmod N$ $M(A,A) = 2^{520} \cdot 2^{520} \cdot 2^{-512} \bmod N \equiv 2^{528} \bmod N$
 i=5 $A = 2^{528} \bmod N$ $M(A,A) = 2^{528} \cdot 2^{528} \cdot 2^{-512} \bmod N \equiv 2^{544} \bmod N$
 i=6 $A = 2^{544} \bmod N$ $M(A,A) = 2^{544} \cdot 2^{544} \cdot 2^{-512} \bmod N \equiv 2^{576} \bmod N$
 i=7 $A = 2^{576} \bmod N$ $M(A,A) = 2^{576} \cdot 2^{576} \cdot 2^{-512} \bmod N \equiv 2^{640} \bmod N$
 i=8 $A = 2^{640} \bmod N$ $M(A,A) = 2^{640} \cdot 2^{640} \cdot 2^{-512} \bmod N \equiv 2^{768} \bmod N$
 i=9 $A = 2^{768} \bmod N$ $M(A,A) = 2^{768} \cdot 2^{768} \cdot 2^{-512} \bmod N \equiv 2^{1024} \bmod N$

そして、最後の結果 (i=9) が求めたい $R^2 \bmod N$ の解 30 しておくためである。

【0069】ここで求めた $R^2 \bmod N$ と M とを乗算した後、この乗算結果をフロー2における X として、フロー2を実行することにより、求めたい $M \cdot R \bmod N$ が得られる。

【0070】前述した通り、フロー2における演算は、 N 及び N' を使用した乗算と、 R を用いた除算で構成されている。従って、この解は、実質的に除算(剰余演算)を行うことなく乗算と加算のみで求めることができる。

【0071】以上、図1のフローチャートにおいてあらかじめ必要とされる $R \bmod N$ 、 $M \cdot R \bmod N$ 、 N' の値が用意できた。

【0072】次に、図1のフローチャートを実行する。

【0073】まず、あらかじめ求めた $R \bmod N$ を初期値としてレジスタA (レジスタは、例えばメモリであっても良い。以下同様である。)に格納する。そして、あらかじめ求めた $M \cdot R \bmod N$ をレジスタBに格納する。 $M \cdot R \bmod N$ をレジスタBに格納する理由は、後の処理で使用する式(25)中の $M \cdot R \bmod N$ を確保

*なわち、 N' 値は運用者から与えられるものである。

【0065】 $[M \cdot R \bmod N]$ について] あらかじめ $R^2 \bmod N$ の値を求めておくと、モンゴメリ演算を使用した以下の計算

$$\equiv M \cdot R \bmod N$$

※【0068】

【0074】次に、式(24)を実行する。すなわちモンゴメリ演算 $A^2 \cdot R' \bmod N$ を実行する。式(24)は、前述したようにフロー2を実行することにより得られる。今、レジスタAに格納された値を2乗した値が式(19)における X として計算され、 m が求められる。前述のようにこの演算は、乗算と R を用いた剰余演算によって実行される。 $R = 2^n$ という定義なので、 R を用いた剰余演算は、被除算値の 2^n 以下の値を見るだけでよい。次に、求められた m を用いて、式(20)を実行する。今、レジスタAに格納された値を2乗した値が式(20)における X として計算され、 t が求められる。前述のようにこの演算は、乗加算と R を用いた除算によって実行される。 $R = 2^n$ という定義なので、 R を用いた除算は、被除算値の 2^n 以上の値を見るだけでよい。そしてこの演算結果をレジスタAに格納する。

【0075】次に、レジスタaに格納された値をレジスタAに格納する。

【0076】ここで、指数 e のビットの判別を行う。もし、ビットが1ならば式(25)を実行する。すなわちモンゴメリ演算 $A \cdot B \cdot R' \bmod N$ を実行する。式

(25)は、前述したようにフロー2を実行することにより得られる。今、レジスタAに格納された値とレジスタBに格納された値との積が式(19)におけるXとして計算され、mが求められる。次に、求められたmを用いて、式(20)を実行する。今、レジスタAに格納された値とレジスタBに格納された値との積が式(20)におけるXとして計算され、tが求められる。そしてこの演算結果をレジスタaに格納する。そして、レジスタaに格納された演算結果をレジスタAに格納する。

【0077】もし指数eのビットが0ならば式(25)の演算を行わずに次のステップに進む。

【0078】次に、指数eの全てのビットに対して以上の演算(式(24)(25)の実行)を行ったかどうか判定し、行っていないのであれば式(24)を実行するステップに戻り、行ったのであれば次に式(26)を実行する。式(26)は、前述したようにフロー2を実行することにより得られる。今、レジスタAに格納された値が式(19)におけるXとして計算され、mが求められる。次に、求められたmを用いて、式(20)を実行する。今、レジスタAに格納された値が式(20)にお

けるXとして計算され、tが求められる。そして、以上の一連の動作が終了する。

【0079】以上のべき乗剰余演算の解法の概略をまとめると以下ようになる。

【0080】

1、べき乗剰余演算($M^e \bmod N$)

↓

2、反復平方積法を用いる。(A・A mod N、A・M mod Nの繰り返し。)

↓

3、A・A mod N、A・M mod Nの計算は、除算が複雑なためモンゴメリ手法に置き換える。(モンゴメリ手法： $M'(X) = X \cdot R' \bmod N$)

↓

$$A^2 \cdot R' \bmod N \cdots (24); A \cdot B \cdot R' \bmod N \cdots (25)$$

(Bは、式(25)における" $M \cdot R \bmod N$ "が位置する部分に相当する。)の2種の演算を、指数"e"のビット値の内容に従って、規定の手順で繰り返し最後に、

$$A \cdot 1 \cdot R' \bmod N \cdots (26)$$

の演算を行えば良いことは、図1及びフロー3で述べたとおりである。

【0086】従って、上記3種類の剰余演算を実行する演算器(以下、コプロセッサと称す。)を用意すれば、そのコプロセッサを用いて(演算の繰り返し手順等は、ソフトウェアあるいはハードウェアのいずれの制御方法で実現しても良い。)、べき乗剰余演算の解を得ることができる。

【0087】本発明は、上記の3種類の剰余演算、すなわち3つの演算モードを有するコプロセッサに関するも

*4、 $M'(X) = X \cdot R' \bmod N$ は、REDC(X)関数をしようして実現できる。(REDC関数： $\bmod N$ の演算を $\bmod R$ の形に変換するもの。これにより除算の複雑性を回避できる。)

次に、本発明を実際に実現するためのハードウェアについて、図面を使用して以下に説明する。

【0081】まず、実際のハードウェアの概略を示すブロック図を図4に示す。

【0082】図4において、401、403、405、407、409、411、413、415は、メモリもしくはレジスタである。そして、401、403、405、407、409、411、413には、各々ブロック図中に記載された値が格納されている。また、レジスタ405は、図1中で説明したレジスタAに相当するものであり、レジスタ415は、レジスタaに相当するものである。セクタ417は、指数eが格納された1ビット左シフトレジスタ409からの指示に基づき、レジスタ407もしくはレジスタ405の出力のいずれかを演算部419中の乗算部に転送する。詳しくは、セクタ417は、図1中の"eシフトキャリ"を実行する部分、すなわち指数部eが1の場合は $M \cdot R \bmod N$ を演算部に転送し、指数部eが0の場合は転送しない。また、図1中のフローの初期段階において、 A^2 を求めるときには、レジスタ405の出力を演算部に転送する。

【0083】演算部419は、乗算部と除算部とで構成されている。この演算部419は、乗算とREDC(X)関数を実行する。

【0084】次に、本発明を実際に実現するためのハードウェアの詳細について、図面を使用して以下に説明する。

【0085】(第1の実施の形態)

(構成)べき乗剰余演算 $M^e \bmod N$ の計算方法は、先に示したモンゴメリ演算手法における、

のである。

【0088】図5は、本発明の第1の実施の形態の剰余演算コプロセッサを示す図である。

【0089】なお、図面中において、各ブロックを結ぶ太い矢印は、データが転送されるバスを示す。

【0090】本発明の剰余演算コプロセッサは、コプロセッサ全体の演算タイミング及び演算器内の各種回路に対して、上記3種類の演算の種類に対応したコントロール信号を供給するタイミング/コントロール回路T/Cと、モンゴメリ法化における各演算値を格納する、複数の演算値メモリSmem、N'mem、Nmem、Mmem、A'mem、Wlmem、Whmemを有する。また、本発明の剰余演算コプロセッサは、乗加算を行う高速乗加算器Mul/Add、加算を行う高速加算器Add、乗算値を格納する乗算値格納レジスタXi-re

g、被乗算値を格納する被乗算値格納レジスタYi-reg、被加算値を格納する被加算値格納レジスタAi-reg、高速加算器Addが出力する値の上位の桁を格納するレジスタRHを有する。

【0091】高速加算器Add、乗算値格納レジスタXi-reg、被乗算値格納レジスタYi-reg、被加算値格納レジスタAi-regは、各演算値メモリから読み出された、高速乗加算器Mul/Addの入力ビット長分に対応した値を一時格納する機能を有する。

【0092】高速乗加算器Mul/Addは、乗算動作の際の入力値として、レジスタXi-regの出力値と、レジスタYi-regの出力値を入力し、加算動作の際の入力値として、レジスタAi-regの出力値を入力する、特定ビット長の乗加算器である。そして、その出力は、次段の高速加算器Addの加算入力値となる。

【0093】高速加算器Addは、高速乗加算器Mul/Addの出力値と、レジスタRHの出力値との加算動作を行う、特定ビット長の加算器である。そして、この高速加算器Addの出力のうち、上位桁はレジスタRHもしくは各演算値メモリに向けて出力され、下位桁は各演算値メモリに向けて出力される。

【0094】(動作) 次に、図5に示す回路の動作を説明する。

【0095】 $[A^2 \cdot R' \bmod N \cdots (24)]$ の実現の方法] Aの値を演算値メモリA'mem、Wlmemの双方に格納し、モンゴメリ法化におけるN'の値を演算値メモリN'memに格納し、Nの値を演算値メモリNmemに格納する。N及びN'の値は、暗号化/復号化のための鍵であるので、データの送信者/受信者に対して暗号システム運用者により決定されている。Rの値は、Nの値から自ずと決定される。Aの値は、前述したように、 $R \bmod N$ を実行することで決定される。

【0096】そして、タイミング/コントロール回路T/Cに対して、モード1信号を供給することにより、以下のように上記 $A^2 \cdot R' \bmod N \cdots (24)$ が実行される。

【0097】まず、 $A \times A$ の計算が行われる。

【0098】演算値メモリA'memから高速乗加算器Mul/Addの入力ビット長分に対応した値が、レジスタXi-regに取り込まれる。同様に、演算値メモリWlmemから高速乗加算器Mul/Addの入力ビット長分に対応した値が、レジスタYi-regに取り込まれる。

【0099】図6に示すように、乗算動作を行う場合に、演算値のビット長(例えば16ビット)が高速乗加算器Mul/Addの乗算入力処理ビット長(例えば4ビット)を越えているとき、演算値のビット長に対応した演算の回数分($4 \times 4 = 16$ 回)だけ単位乗加算を繰り返すことになる。

【0100】上述の動作をハードウェアイメージにより詳細に表現すると図7ようになる。

【0101】この演算は、演算値メモリA'memの各アドレスA'3、A'2、A'1、A'0に格納された値と演算値メモリWlmemの各アドレスWl3、Wl2、Wl1、Wl0に格納された値とを乗算したときの値が、演算値メモリWhem及び演算値メモリWlmemの各アドレスWh3、Wh2、Wh1、Wh0、Wl3、Wl2、Wl1、Wl0に格納されることを意味する。

【0102】(単位乗加算1) まず、レジスタYi-regが、演算値メモリWlmemのアドレスWl0に格納された値(被乗算値)を取り込むとともに、レジスタXi-regが演算値メモリA'memのアドレスA'0に格納された値(乗算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、高速加算器Addに向けて出力する。この単位乗加算1では、桁合わせ動作を行う必要がないため、レジスタAi-regから高速乗加算器Mul/Addへ与える加算値の値を0とする。(レジスタAi-regの値を0とするか、レジスタAi-regに格納された値が高速乗加算器Mul/Addへ与えられないようにする。)さらに、高速加算器AddのThrough端子に"highレベル"を供給して、レジスタRHの内容を加算しないようにする。そして、演算値メモリA'memの値と演算値メモリWlmemの値との乗算において、単位乗加算1の演算結果の下位桁は、本乗算の最終演算結果の最下位桁であるので、この単位乗加算1の演算結果の下位桁は本乗算の最終演算結果として演算値メモリWlmemの最下位アドレスWl0に格納される。

(図7において、アンダーラインが付与されたアドレスには、最終演算結果が格納されていることを意味する。)この単位乗加算1の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0103】(単位乗加算2) レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリA'memのアドレスA'1に格納された値(乗算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、高速加算器Addに向けて出力する。この単位乗加算2では、レジスタAi-regから高速乗加算器Mul/Addへ与える加算値の値を0とする。(レジスタAi-regの値を0とするか、レジスタAi-regに格納された値が高速乗加算器Mul/Addへ与えられないようにする。)が、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算1の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算2の演算結果の下位桁は、演算値メモリWhmem

の最下位アドレスWh0に格納される。この単位乗加算2の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0104】(単位乗加算3) レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリA'memのアドレスA'2に格納された値(乗算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、高速加算器Addに向けて出力する。この単位乗加算3では、レジスタAi-regから高速乗加算器Mul/Addへ与える加算値の値を0とする。(レジスタAi-regの値を0とするか、レジスタAi-regに格納された値が高速乗加算器Mul/Addへ与えられないようにする。)が、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算1の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算3の演算結果の下位桁は、演算値メモリWhmemのアドレスWh1に格納される。この単位乗加算3の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0105】(単位乗加算4) レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリA'memのアドレスA'3に格納された値(乗算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、高速加算器Addに向けて出力する。この単位乗加算4では、レジスタAi-regから高速乗加算器Mul/Addへ与える加算値の値を0とする。(レジスタAi-regの値を0とするか、レジスタAi-regに格納された値が高速乗加算器Mul/Addへ与えられないようにする。)が、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算1の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算4の演算結果の下位桁は、演算値メモリWhmemのアドレスWh2に格納される。この単位乗加算4の演算結果の上位桁は、次の単位乗加算の桁合わせのために演算値メモリWhmemのアドレスWh3に格納される。

【0106】(単位乗加算5) レジスタYi-regは、演算値メモリWlmemのアドレスWl1に格納された値(被乗算値)を取り込み、レジスタXi-regは、演算値メモリA'memのアドレスA'0に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh0に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算5では、高速加算器AddのTh

rough端子に"highレベル"を供給して、レジスタRHの内容(単位乗加算4の演算結果の上位桁)を加算しないようにする。この単位乗加算5の演算結果の下位桁は、演算値メモリWlmemのアドレスWl1に格納される。この単位乗加算5の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0107】(単位乗加算6) レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリA'memのアドレスA'1に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh1に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算6では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算5の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算6の演算結果の下位桁は、演算値メモリWhmemのアドレスWh0に格納される。この単位乗加算6の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0108】(単位乗加算7) レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリA'memのアドレスA'2に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh2に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算7では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算6の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算7の演算結果の下位桁は、演算値メモリWhmemのアドレスWh1に格納される。この単位乗加算7の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0109】(単位乗加算8) レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリA'memのアドレスA'3に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh3に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算8では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算7の演算結果の

上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算8の演算結果の下位桁は、演算値メモリWhmemのアドレスWh2に格納される。この単位乗加算8の演算結果の上位桁は、演算値メモリWhmemのアドレスWh3に格納される。

【0110】単位乗加算9から16までは、前述の単位乗加算に習って、図7のごとく実行される。そして、 $A \times A$ すなわち、演算値メモリA'memに格納された値と演算値メモリWlmemに格納された値とを乗算したときの値が、Whmem-Wlmem("ー"は、引き算を示すものではない。)の形で格納される。

【0111】例えば、8ビット×8ビットの計算、 $64H \times F5H = 6572H$ を例にとると、計算結果6572Hについて、上位桁65Hを演算値メモリWhmemに、下位桁72Hを演算値メモリWlmemにそれぞれ格納する。

【0112】次に、REDCC関数を使用して、モンゴメリ手法を用いた剰余演算を実行する。モンゴメリ手法において、オペランド長nビットのべき乗剰余演算を実行するとき、 $R = 2^n$ とすることは前述した通りである。そして、この関係でのnビット長の R' 、 N' の値は存在する。そして、 $A \times A$ の値は、Whmem-Wlmem*

$$t = (X + m \cdot N) / R$$

$$= [(Whmem - Wlmem) + (Mmem) \cdot (Nmem)] / R$$

上式の解を得るためには、まず $(Mmem) \times (Nmem)$ の計算を先の $A \times A$ に習って実行し、その結果を

$$Mmem \times 1 + Wlmem \cdots (27) \quad (\text{下位桁の足し算})$$

$$A'mem \times 1 + Whmem \cdots (28) \quad (\text{上位桁の足し算})$$

の計算を実行する。このときの式(28)が求めるtである。

【0118】また、R以下の値は必ず0であるから、式(27)の計算結果0となるので、メモリに格納する必要はない。なお、式(27)の計算では、桁あふれが発生する場合が考えられるが、そのときには、式(28)の計算の際に、図5に示した高速乗加算器Mul/Addの+1端子に、1を供給する。これらの計算は、本コプロセッサを用いれば容易に実行することができるのは★

$$[(\text{反転}Nmem + 1) \times 1] + Wlmem \cdots (29)$$

の計算を行って求める。反転Nmemは、Nmemに格納された値のインバースを意味する。

【0121】ここで、演算値メモリNmemに格納されているNの値は、暗号アルゴリズムの世界では奇数であるから、式(29)の(反転Nmem+1)は"Nmemの1の補数を取り、最下位ビットを1とする"ことで求められる。この動作は、図5に示すレジスタXi-regのInv端子に1を供給する(レジスタXi-regに格納された値の1の補数をとる。)ことで実現される。

【0122】 $[A \cdot B \cdot R' \bmod N \cdots (25)]$ の実現の方法] Aの値を演算値メモリWlmemに、Bの

*mの形で格納されているので、mの計算は以下になる。

【0113】

$$m = (X \bmod R) \cdot N' \bmod R$$

$$= (Whmem - Wlmem) \bmod R \cdot N' \bmod R$$

ここで、 $(X \bmod R)$ は、XをRで割った余りをしめすので、R以下の値を見ればよいので、

$$= (Wlmem) \cdot (N'mem) \bmod R$$

となる。

【0114】そして、上式は $(Wlmem) \cdot (N'mem)$ をRで割った余りを示すので、 $(Wlmem) \cdot (N'mem)$ を実行した結果の下位nビットの値がmとなる。

【0115】すなわち、 $(Wlmem) \cdot (N'mem)$ の計算を、上述の $A \times A$ の計算に習って実行し、その結果をA'mem-Mmemの形で、演算値メモリA'memと演算値メモリMmemの双方に格納したときの、演算値メモリMmemに格納された値がmである。

【0116】また、tの計算は以下になる。

【0117】

★明かである。

【0119】式(28)で得られた値は、演算値メモリA'memとMmemの双方に格納する。また、桁あふれが発生した場合は、キャリーフラグCFに1を書き込む。

【0120】ところで、キャリーフラグが1の場合は、 $t > N$ の関係にあることは明かであるから、 $t - N$ の計算を行う必要がある。これは、

値を演算値メモリSmemに格納し、モンゴメリ法化における N' の値を演算値メモリN'memに格納し、Nの値を演算値メモリNmemに格納する。

【0123】そして、タイミング/コントロール回路T/Cに対して、モード2信号を供給することにより、以下のように上記 $A \cdot B \cdot R' \bmod N \cdots (25)$ が実行される。

【0124】まず、 $A \times B$ の計算を行う。

【0125】演算値メモリSmemから高速乗加算器Mul/Addの入力ビット長分に対応した値が、レジスタXi-regに取り込まれる。同様に、演算値メモリWlmemから高速乗加算器Mul/Addの入力ビッ

40

ト長分に対応した値が、レジスタYi-regに取り込まれる。

【0126】このA×Bの計算は、レジスタXi-regへの値の取り込み元が演算値メモリSmemに変わるだけで、前述のA×Aの計算と同様である。

【0127】次に、REDC関数を使用して、モンゴメリ手法を用いた剰余演算を実行する。

【0128】計算の方法は、A×Aの場合と同様であり、計算結果は、演算値メモリA'memとWlmemの双方に格納される。

【0129】べき乗剰余演算を行う場合は、式(24)と式(25)を繰り返し計算することになる。しかし、以上説明した各計算の実現方法を用いれば、式(24)(25)の計算結果は、ともに演算値メモリA'memとWlmemの双方に格納されるため、各々の式の計算の開始時に、演算値の初期設定等を行う必要がないので、繰り返し計算をスムーズに行うことができる。

【0130】[A・R' mod N・・・(26)の実現*

$$t = (X + m \cdot N) / R$$

$$= [(Wlmem) \times (Nmem) \text{ の結果の } R \text{ 以上の値}] + 1 \cdots (30)$$

式(26)の計算では、上記のXはAであり、R以上の値はない(0と考えてよい)ので、式(24)の実現方法で説明したようなR以上の値(Whmem)の加算は行う必要がない。また(X+m・N)は、必ずRの倍数になるのであるから、式(30)の第2辺に示されるXの値を加算する必要もなく、代わりに"[(Wlmem)×(Nmem)の結果のR以上の値]+1"で解が得られる。これは、暗号アルゴリズムの世界で剰余演算を使用するとき、本計算におけるAの値が0になることがないからである。

【0136】従って、式(30)では、まず(Wlmem)×(Nmem)の計算を実行して、その結果をA'mem-Wlmemの形で、演算値メモリA'memとWlmemに格納すればよい。その結果の得られたA'mem上の値が求める解となる。(本演算の実行でべき乗剰余演算のすべての演算は終了するため、ここでは演算値メモリWlmemへの格納を省略してもよい。)なお、このとき計算の最終結果を演算値メモリA'memの最下位桁に格納するタイミングで、図5に示した+1端子に1を供給して、式(30)の1を足す動作を実行する。

【0137】以上のように、第1の実施形態によれば、規定ビット長の乗加算器(乗算器と加算器を別個に設けてもよい)を演算部の中心として構成し、その周辺に設けられた回路に、タイミング/コントロール信号発生回路からの制御信号を供給することにより、モンゴメリ手法を用いた多ビット長の剰余演算あるいはべき乗剰余演算を実現できる。また、この方式によれば、演算部の中心が限られた(規定の)ビット長の演算器で構成できる

*の方法] Aの値を演算値メモリWlmemに格納し、モンゴメリ法化におけるN'の値を演算値メモリN'memに格納し、Nの値を演算値メモリNmemに格納する。

【0131】そして、タイミング/コントロール回路T/Cに対して、モード3信号を供給することにより、以下のように上記A・R' mod N・・・(26)が実行される。

【0132】今回の式では、A×?という乗算を実行する必要がない。従って、モンゴメリ手法を用いた剰余演算のみを実行する。

【0133】剰余演算におけるmの計算の方法は、A×Aの場合とほぼ同様であり、計算結果は、A'mem-Wlmemの形で、演算値メモリA'memとWlmemの双方に格納したときの演算値メモリWlmemの値がmの値となる。

【0134】また、tの計算は以下になる。

【0135】

ため回路規模が小さくてすみ、LSI化に適している。

【0138】(第2の実施形態)

(構成) 図8は、本発明の第2の実施の形態の剰余演算コプロセッサを示す図である。

【0139】第2の実施形態は、第1の実施形態の構成(図5)に、演算値が0であることを検出して、次の演算のシーケンスを制御する回路を追加したものである。

【0140】具体的にこの制御回路は、図8における制御回路ZeroCである。この制御回路ZeroCは、演算値が0であることを検出して、次演算のシーケンスを制御する回路であり、高速加算器Addの出力信号が与えられる入力端子と出力端子とを有している。この出力端子からは、タイミング/コントロール回路T/Cで発生するタイミング/コントロール信号及びコプロセッサ内の各種回路を動作させる信号に所定の制御を施す信号が出力する。

【0141】(動作) 第1の実施形態の式(24)(25)の実現方法で述べた、REDC関数の計算において、前の乗算の結果得られたXの値は次の2つの場合分けができる。

【0142】(a) Rの倍数である場合・・・前計算で得られた値(Wlmem)が0。

【0143】(b) Rの倍数でない場合・・・前計算で得られた値(Wlmem)が0でない。

【0144】本実施の形態では、制御回路ZeroCが、上記(a)(b)のように、X値(A×AやA×Bの結果の値)のR未満の値が0か否かを検出して、その結果に応じて、続く演算のシーケンスを次の通りに実行させる。

【0145】(a)の場合は、 $X \bmod R = 0$ なので $m = 0$ となるのは明かであるから、このときは m の計算を行う必要はない。

$$t = (X + m \cdot N) / R = X / R = \text{Whmem} \cdots (31)$$

これは、 $A \times A$ 及び $A \times B$ の計算結果の上位桁がそのまま t の値となっていることを示す。従って、この場合のハードウェア処理は、演算値メモリ Whmem に格納された値をそのまま演算値メモリ $A' \text{ mem}$ と Wlmem の双方に格納し、次の繰り返し演算に備える。すなわち、 X 値(前計算で得られた値)の R 未満の値が0の場合、 m 、 t の演算の必要がない。このことは、 t の計算において、 X の全ビット長分を計算する必要がないことを意味する。

$$\begin{aligned} t &= (X + m \cdot N) / R \\ &= \text{Whmem} + [(\text{Wlmem}) \times (\text{Nmem}) \text{の結果の} R \text{以上の値}] + 1 \\ &\cdots (32) \end{aligned}$$

式(32)は、まず $(\text{Wlmem}) \times (\text{Nmem})$ の計算を実行して、その結果を $A' \text{ mem} - \text{Wlmem}$ の形★

$$(A' \text{ mem}) \times 1 + (\text{Whmem}) + 1 \cdots (33)$$

の計算を実行すれば解が得られる。なぜなら、 $X + m \cdot N$ の値は、必ず R の倍数になるから、 R 未満の値をわざわざ計算対象にする必要がないからである。なお、式(33)では、最下位桁の計算タイミングで、図5に示した+1端子に1を供給して式(33)の1を足す動作を実行する。

【0151】式(33)で得られた計算結果は、演算値メモリ $A' \text{ mem}$ と Wlmem の双方に格納するとともに、桁あふれが発生した場合は、キャリーフラグ CF に1を書き込む。その後の処理は、第1の実施形態と同様である。

【0152】以上のように、第2の実施形態によれば、演算値が0であることを検出して、次演算のシーケンスを制御する制御回路(乗加算器のビット長に相当する回路規模で済むので、小規模な回路である。)を付加することにより、第1の実施形態で説明したコプロセッサの演算値メモリ Mmem の削除と、演算量の削減が達成できる。従って、ハードウェアの削減と演算時間の短縮化を図ることができる。

【0153】(第3の実施形態)

(構成) 図9は、本発明の第3の実施形態の剰余演算コプロセッサを示す図である。

【0154】第3の実施形態は、第1の実施形態の構成(図5)もしくは第2の実施形態の構成(図8)に、オペランドビット長を選択してタイミング/コントロール信号を変化させるビット長選択制御回路を付加したものである。

【0155】具体的にこのビット長選択制御回路は、図9における LenCont である。このビット長選択制御回路 LenCont は、入力信号 Sel-len に従って、タイミング/コントロール回路 T/C で発生する

* 【0146】そして t の計算は次のようになる。
【0147】

※ 【0148】(b)の場合は、 m を求めるための計算を第1の実施形態の通り実行し、計算結果を $A' \text{ mem} - \text{Wlmem}$ の形で演算値メモリ $A' \text{ mem}$ と Wlmem の双方に格納し、演算値メモリ Wlmem に格納された値を m の値とする。そして、引き続き t の計算を実行する。

【0149】そして、 t の計算は次のようになる。

【0150】

★で演算値メモリ $A' \text{ mem}$ と Wlmem の双方に格納し、続いて

演算タイミング信号とコプロセッサ内の各種回路に供給するコントロール信号を制御するように動作する。

【0156】(動作) さて、本コプロセッサの動作は、オペランドビット長の変化に伴い、 R 値及びその値により決まる R' 値、 N' 値のビット長と、規定ビット長乗加算の繰り返し手順(回数)とが変化するようになる。

【0157】例えば、高速乗加算器 Mul/Add が16ビット長の時、オペランドビット長512ビットの $A \times A$ の乗算を実行する場合、高速乗加算器 Mul/Add による乗加算の繰り返し回数は次のようになる。

30 【0158】

$$(512/16) \times (512/16) = 1024 \text{ 回}$$

一方、オペランドビット長768ビットの場合は次のようになる。

【0159】

$$(768/16) \times (768/16) = 2304 \text{ 回}$$

そして、これらの計算のシーケンスもそれぞれ異なる。また、前に述べたようにオペランドビット長により R 値は一義的に決まるので、 R' 値、 N' 値のビット長もそれに応じて変化する。

40 【0160】ビット長選択制御回路 LenCont は、これらの変化要因を制御する、すなわち、選択されたビット長に対応して、タイミング/コントロール回路 T/C で発生させる演算タイミング信号やコントロール信号が出力するように制御する。

【0161】一般的に、ビット長選択制御回路 LenCont は、PLAや論理回路等の比較的小規模な回路構成で実現できる。

50 【0162】以上のように、第3の実施形態のコプロセッサでは、ビット長選択制御回路 LenCont を付加することによって、さまざまなオペランドビット長の剰

余演算あるいはべき乗剰余演算を実行することが可能となる。

【0163】（第4の実施形態）

（構成）以上の実施例で示したコプロセッサの演算の基本は、規定ビット長の乗加算である。この基本乗加算を繰り返して剰余演算あるいはべき乗剰余演算を実現する方法は、以上の実施例で示した通りである。

【0164】しかしながら、以上の実施例では、剰余演算モードのみを行うコプロセッサであったため、演算の基本が乗加算であるにも関わらず、その用途が剰余演算に限られている。

【0165】そこで、さまざまな演算の基本となる多ビット長乗加算のモードを追加して、本コプロセッサの汎用性を向上させる。

【0166】図10は、本発明の第4の実施の形態の剰余演算コプロセッサを示す図である。

【0167】第4の実施の形態は、先の実施形態に乗加算のモードを追加したものであり、このモードを実行するためにタイミング／コントロール回路T/Cにモード信号4を供給するようにしたものである。

【0168】（動作）以下に示す乗加算を実行する例を以下に説明する。

【0169】 $A \cdot B + C \cdots (34)$

まず、式(34)において、値Aを演算値メモリWlmemに、値Bを演算値メモリSmemに、値Cを演算値メモリWhmemに格納する。そして、タイミング／コントロール回路T/Cにモード信号4を供給して、演算の種類を、乗加算を実行するモードにする。

【0170】上述の動作をハードウェアイメージでより詳細に表現すると図11のようになる。

【0171】この演算は、演算値メモリSmemの各アドレスS3、S2、S1、S0に格納された値と演算値メモリWlmemの各アドレスWl3、Wl2、Wl1、Wl0に格納された値とを乗算し、この結果と、演算値メモリWhmemの各アドレスWh3、Wh2、Wh1、Wh0に格納された値とを足し合わせたときの値が、演算値メモリWhmem及び演算値メモリWlmemの各アドレスWh3、Wh2、Wh1、Wh0、Wl3、Wl2、Wl1、Wl0に格納されることを意味する。

【0172】（単位乗加算1）まず、レジスタYi-regが、演算値メモリWlmemのアドレスWl0に格納された値（被乗算値）を取り込むとともに、レジスタXi-regが演算値メモリSmemのアドレスS0に格納された値（乗算値）を取り込み、レジスタAi-regが演算値メモリWhmemのアドレスWh0に格納された値（加算値）を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、かつこの乗算結果に加算値を加算し高速加算器Addに向けて出力する。この単位乗加算1では、高速加算器Addの

Through端子に”highレベル”を供給して、レジスタRHの内容を加算しないようにする。

【0173】そして、この単位乗加算における演算結果の下位桁は、本乗加算の最終演算結果の最下位桁であるので、この単位乗加算1の演算結果の下位桁は本乗算の最終演算結果として演算値メモリWlmemの最下位アドレスWl0に格納される。（図11において、アンダーラインが付与されたアドレスには、最終演算結果が格納されていることを意味する。）この単位乗加算1の演算結果の上位桁は、次の単位乗加算のためにレジスタRHに格納される。

【0174】（単位乗加算2）レジスタYi-regは、すでに格納された値（被乗算値）を維持し、レジスタXi-regは、演算値メモリSmemのアドレスS1に格納された値（乗算値）を取り込み、レジスタAi-regが演算値メモリWhmemのアドレスWh1に格納された値（加算値）を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、かつこの乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算2では、高速加算器AddのThrough端子に”lowレベル”を供給して、レジスタRHの内容（単位乗加算1の演算結果の上位桁）を高速乗加算器Mul/Addの出力に加算する。この単位乗加算2の演算結果の下位桁は、演算値メモリWhmemの最下位アドレスWh0に格納される。この単位乗加算2の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0175】（単位乗加算3）レジスタYi-regは、すでに格納された値（被乗算値）を維持し、レジスタXi-regは、演算値メモリSmemのアドレスS2に格納された値（乗算値）を取り込み、レジスタAi-regが演算値メモリWhmemのアドレスWh2に格納された値（加算値）を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、かつこの乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算3では、高速加算器AddのThrough端子に”lowレベル”を供給して、レジスタRHの内容（単位乗加算2の演算結果の上位桁）を高速乗加算器Mul/Addの出力に加算する。この単位乗加算3の演算結果の下位桁は、演算値メモリWhmemの最下位アドレスWh1に格納される。この単位乗加算3の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0176】（単位乗加算4）レジスタYi-regは、すでに格納された値（被乗算値）を維持し、レジスタXi-regは、演算値メモリSmemのアドレスS3に格納された値（乗算値）を取り込み、レジスタAi-regが演算値メモリWhmemのアドレスWh3に格納された値（加算値）を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、か

つこの乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算4では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算3の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算4の演算結果の下位桁は、演算値メモリWhmemの最下位アドレスWh2に格納される。この単位乗加算4の演算結果の上位桁は、次の単位乗加算の桁合わせのために演算値メモリWhmemのアドレスWh3に格納される。

【0177】(単位乗加算5)レジスタYi-regは、演算値メモリWlmemのアドレスWl1に格納された値(被乗算値)を取り込み、レジスタXi-regは、演算値メモリSmemのアドレスS0に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh0に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算5では、高速加算器AddのThrough端子に"highレベル"を供給して、レジスタRHの内容(単位乗加算4の演算結果の上位桁)を加算しないようにする。この単位乗加算5の演算結果の下位桁は、演算値メモリWlmemのアドレスWl1に格納される。この単位乗加算5の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0178】(単位乗加算6)レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリSmemのアドレスS1に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh1に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算6では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算5の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算6の演算結果の下位桁は、演算値メモリWhmemのアドレスWh0に格納される。この単位乗加算6の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

【0179】(単位乗加算7)レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリSmemのアドレスS2に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh2に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、

その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算7では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算6の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算7の演算結果の下位桁は、演算値メモリWhmemのアドレスWh1に格納される。この単位乗加算7の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタRHに格納される。

10 【0180】(単位乗加算8)レジスタYi-regは、すでに格納された値(被乗算値)を維持し、レジスタXi-regは、演算値メモリSmemのアドレスS3に格納された値(乗算値)を取り込み、レジスタAi-regは、演算値メモリWhmemのアドレスWh3に格納された値(加算値)を取り込む。次に、高速乗加算器Mul/Addはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Addに向けて出力する。この単位乗加算8では、高速加算器AddのThrough端子に"lowレベル"を供給して、レジスタRHの内容(単位乗加算7の演算結果の上位桁)を高速乗加算器Mul/Addの出力に加算する。この単位乗加算8の演算結果の下位桁は、演算値メモリWhmemのアドレスWh2に格納される。この単位乗加算8の演算結果の上位桁は、演算値メモリWhmemのアドレスWh3に格納される。

20 【0181】単位乗加算9から16までは、前述の単位乗加算に習って、図11のごとく実行される。そして、最終演算結果は、Whmem-Wlmem("—"は、引き算を示すものではない。)の形で格納される。

30 【0182】図12には、 $8591 \times 4673 + 2069 = 40147812$ を、上述のハードウェアで実行した場合の単位乗加算を示す。動作の説明については、図11と同様であるので省略する。以上のように、第4の実施形態によれば、さまざまな演算の基本となる多ビット長の乗加算のモードを実現できるので、コプロセッサの汎用性を大幅に向上させることができる。

【0183】(第5の実施形態)

(構成)以上の実施の形態で示したコプロセッサは、高速乗加算器Mul/Addと高速加算器Addの入力ビット長で決まる単位乗加算毎に、メモリへのアクセスを行う必要があるため、演算時間の短縮化においては改善の余地がある。なぜなら、メモリを含んだ回路全体の動作速度は、メモリアクセスタイムが回路全体の動作速度を制限してしまうからである。

【0184】そこで、本実施の形態では、単位乗加算の演算回数に対するメモリのアクセス回数を減らして、最終的な演算時間の短縮を図る。

【0185】図13は、本発明の第5の実施の形態の剰余演算コプロセッサを示す図である。

50 【0186】第5の実施の形態は、先の実施の形態の構

成に対して、被乗算値格納レジスタYi-regを複数個に増やして、その出力を選択する回路を設けたものである。さらに、高速加算器Addの上位桁あるいは下位桁の出力値を格納するレジスタも被乗算値格納レジスタの個数に対応する個数設け、各レジスタの出力値を適当に選択する選択回路を設けたものである。

【0187】具体的には、レジスタYi-reg [0]、レジスタYi-reg [1]、レジスタYi-reg [2]、レジスタYi-reg [3]が、高速乗加算器Mul/Addの入力ビット長に対応した長さを持つ被乗算値レジスタであり、各々の出力は、被乗算値選択回路Yi-selに接続されている。選択回路Yi-selは、レジスタYi-reg [0]、レジスタYi-reg [1]、レジスタYi-reg [2]、レジスタYi-reg [3]の値のうちのいずれか1つを選択して、後続の高速乗加算器Mul/Addに与える選択回路である。

【0188】レジスタRAは、高速加算器Addの出力のうち上位桁R-highを一時格納するレジスタであり、その出力は後続の選択回路SelA、SelB、EnSelに接続されている。そして高速加算器Add出力のうち下位桁R-lowは、選択回路SelB及びイネーブルバッファEnに接続されている。

【0189】レジスタAi-regの出力は、選択回路SelAに接続されている。そして選択回路SelAは、レジスタAi-regとレジスタRAの内容のうちのいずれかを選択して高速加算器Addに加算入力すべく加算加算器Addに接続されている。

【0190】選択回路SelBは、R-lowとレジスタRAの内容のうちのいずれかを選択してレジスタRBに向けてその結果を出力すべくレジスタRBに接続されている。

【0191】レジスタRBは、選択回路SelBの出力を一時格納するレジスタであり、出力はレジスタRC及びEnselに接続されている。

【0192】レジスタRCは、レジスタRBの出力を一時格納するレジスタであり、出力はレジスタRD及びEnselに接続されている。

【0193】レジスタRDは、レジスタRCの出力を一時格納するレジスタであり、出力は選択回路Ensel及び高速乗加算器Mul/Addに接続されている。

【0194】選択回路Enselは、レジスタRA、RB、RC、RDの内容のうちのいずれか1つを選択して演算値メモリに向けて出力する選択回路である。

【0195】その他の回路については、先の実施の形態と同様な働きをするので説明を省略する。

【0196】(動作) 次に、演算の基本となる

$$(A3A2A1A0) \times S0 + C0$$

の計算方法について、

$$(W13, W12, W11, W10) \times S0 = (RA,$$

RB, RC, RD, W10)

のようなハードウェアイメージを使用して、図14を用いて説明する。

【0197】なお、アンダーラインが付与されたアドレスは、演算の最終結果が格納されていることを意味する。

【0198】(初期設定) まず図示のごとく初期設定が行われる。

【0199】(ステップ1) レジスタXi-regの値と、選択回路Yi-selで選択されたレジスタYi-reg [0]の値と、レジスタRDの値と選択回路SelAで選択されたレジスタAi-regの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるR-highは、レジスタRAに、演算結果の下位桁であるR-lowは、演算値メモリW1memに格納される。同時に、レジスタRCの値はレジスタRDに、レジスタRBの値はレジスタRCに、レジスタRAの値は選択回路SelBで選択されたレジスタRBに格納される。

(ステップ2) レジスタXi-regの値と、選択回路Yi-selで選択されたレジスタYi-reg [1]の値と、レジスタRDの値と選択回路SelAで選択されたレジスタRAの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるR-highは、レジスタRAに、演算結果の下位桁であるR-lowは、選択回路SelBで選択されたレジスタRBに格納される。同時に、レジスタRCの値はレジスタRDに、レジスタRBの値はレジスタRCに格納される。

【0200】(ステップ3) レジスタXi-regの値と、選択回路Yi-selで選択されたレジスタYi-reg [2]の値と、レジスタRDの値と選択回路SelAで選択されたレジスタRAの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるR-highは、レジスタRAに、演算結果の下位桁であるR-lowは、選択回路SelBで選択されたレジスタRBに格納される。同時に、レジスタRCの値はレジスタRDに、レジスタRBの値はレジスタRCに格納される。

【0201】(ステップ4) レジスタXi-regの値と、選択回路Yi-selで選択されたレジスタYi-reg [3]の値と、レジスタRDの値と選択回路SelAで選択されたレジスタRAの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるR-highは、レジスタRAに、演算結果の下位桁であるR-lowは、選択回路SelBで選択されたレジスタRBに格納される。同時に、レジスタRCの値はレジスタRDに、レジスタRBの値はレジスタRCに格納される。

【0202】以上のステップ1からステップ4までの処

理では、演算値メモリへのアクセスは、ステップ1で1度行っているだけである。従って、残りのステップ2からステップ4までの期間に、各演算値メモリのアドレスの変更、プリチャージ等を行ってメモリアクセスの時間をかせぐことができる。また、ステップ2からステップ4までの期間内に行う動作は、ほとんど同じであるため、演算処理が複雑になることはない。

【0203】次に、他の演算の例として、

$(A3, A2, A1, A0) \times (B3, B2, B1, B0) + (C3, C2, C1, C0)$

の計算方法について、

$(W13, W12, W11, W10) \times (S3, S2, S1, S0) + (Wh3, Wh2, Wh1, Wh0) = (RA, RB, RC, RD, W13, W12, W11, W10)$

のようなハードウェアイメージを使用して、図15、図16に示す。

【0204】さらに、実際の計算の例として、図17に $8591 \times 4673 + 2069 = 40147812$ のタイム1からタイム4までの演算過程を示す。

【0205】なお、アンダーラインが付与された値は、演算の最終結果であることを意味する。

【0206】この図から、最終結果の上位の桁は、タイム4のステップ4で得られたレジスタRA、RB、RC、RDに格納された値であり、下位の桁はステップ1で得られた演算値メモリW1memの値であることが理解できる。

【0207】以上のように第5の実施形態によれば、乗加算器等の演算処理部のサイズを変える必要はなく、全体的に小規模な回路で演算時間の短い剰余演算コプロセッサを実現できる。

【0208】(第6の実施形態)

(構成) 図18は、本発明の第6の実施の形態を示すブロック図であり、先に説明したコプロセッサと外部装置との間に、演算値メモリインタフェース回路と演算コントロール回路を設けたものである。

【0209】(演算値メモリインタフェース回路MemIF) MemIFは、MCUとコプロセッサ内の演算値メモリとの間でデータの授受を行うための演算値メモリインタフェース回路である。

【0210】演算値メモリは、演算に先だってコプロセッサ外部から演算データを格納し、演算の終了時に演算結果の値をコプロセッサ外部に送出する。同時に、演算実行中はコプロセッサ外部とは関係なくダイナミックに演算ユニットとのアクセスを繰り返す。つまり、演算値メモリは2種類の通信プロトコルを有する。演算値メモリインタフェースMemIFでは、それを実現すべく回路を構成する。

【0211】MCUから出力されるアドレス信号adsとメモリ制御信号Memconは、演算値メモリイン

タフェイスMemIFに入力され、演算値メモリインタフェースMemIF内で、コプロセッサ内の各演算値メモリ別のアドレス信号Comemadとメモリコントロール信号Comconを作成してコプロセッサに供給する。MDbusは、MCUのデータバスであり、CoDbusはコプロセッサの外部インタフェース用データバスである。

【0212】ここで、各演算値メモリがMCU内のある単一のメモリ空間上に配置されている場合は、演算値メモリインタフェースMemIFに入力されるads信号とMemcon信号とMDbusは一種類でよいが、複数のメモリ空間上に配置されている場合は複数種類の入力となる。また、一般的にはMDbusとCoDbusは直結されることが多いが、例えばコプロセッサ内部で処理される演算値メモリのデータ長とMCUのデータ長が異なる場合などは、演算値メモリインタフェースMemIFを介してデータ変換を行うこともできる。

【0213】なお、コプロセッサが演算実行中は、演算値メモリが演算実行のためにダイナミックに動作しているので、MCUからのアクセスを禁止するように制御されている。このようにして、コプロセッサ内演算値メモリと外部装置との第1の通信プロトコルを実現する。

【0214】次に、演算実行の際、演算値メモリインタフェースMemIFにはコプロセッサ内のタイミング/コントロール回路T/Cから出力される演算実行時メモリコントロール信号Exemcが入力され、それを受け取った演算値メモリインタフェースMemIFは、演算値メモリが演算ユニットとの間でデータの授受を行えるように処理されたComemad信号とComcon信号をコプロセッサに向けて出力する。このようにして、コプロセッサ内の演算実行中の演算値メモリと演算ユニットとの第2の通信プロトコルを実現する。

【0215】(演算コントロール回路CopCon) CopConは、MCUから出力されるコプロセッサ制御信号Exconを受け取って、コプロセッサ内部へ演算制御信号Sevex(以上の実施の形態で言えば、演算モード信号、ビット長選択信号等)を供給する演算コントロール回路である。ここで、コプロセッサへの演算の開始を促すのは、演算モード信号やコプロセッサ原振クロックCOPCLKを供給することで実現できる。

【0216】また、演算の終了をMCU側で確認するためには、コプロセッサ内のタイミング/コントロール回路T/Cで発生される演算終了のタイミング信号CoendをCopConで受取り、ラッチ回路などで処理した演算終了モニター信号EndmoniをMCUに向けて出力する。

【0217】一般に、演算コントロール回路CopConは、MCUのペリフェラル回路としてローカルメモリエリア等に割り付けてMCUの命令で直接アクセスできるように構成でき、比較的簡単な回路で実現できる。

【0218】なお、図中のMCUCLK信号は、MCUの原振クロックを、COPCLK信号はコプロセッサの原振クロックを示している。

【0219】以上のように第6の実施の形態によれば、比較的の小規模な回路で外部装置（例えば、MCU等）とコプロセッサとのインタフェースが実現できるので、外部インタフェース付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内蔵のMCUを構成でき、しかもLSI化が可能となる。

【0220】（第7の実施形態）第6の実施の形態では、演算終了をコプロセッサ外部で確認する手段は、演算終了モニター信号Endmoniを見るしかない。しかしこの方法では、コプロセッサで処理される多ビット長の剰余演算量は多いため、例えば外部装置がMCUの場合はMCU側で常にEndmoni信号を監視する時間が比較的長くなり、MCUの動作パフォーマンスが低下する。

【0221】本実施の形態はこの問題を解決するために、専用の演算終了時の割り込み制御回路を設けたものである。

【0222】図19は、本発明の第7の実施の形態を示すブロック図である。

【0223】図19中で、IntConは演算終了割り込み制御回路であり、あらかじめMCUから出力される割り込み設定信号Intsetにより、割り込みの準備がなされている。コプロセッサより演算終了のタイミング信号Coendが入力されると、CopConで設定された演算モードの種類別に、割り込みに必要な割り込み処理リクエスト信号、アクノリッジ信号、ベクター制御信号等をIntsigとして、MCUとの間でやりとりし、最後に割り込みの準備を解除して割り込み処理を終了する。

【0224】なお、演算終了の割り込み要因は1つに固定しても良いが、本コプロセッサは複数の演算モードを有しているため、各演算モード毎の割り込みを行った方が外部装置によるべき乗剰余演算への展開の仕方が容易になるので、割り込み要因を複数に設定したほうが良い。

【0225】Intconは、演算コントロール回路CopConと同様にMCUのペリフェラル回路としてローカルメモリエリア等に割り付けてMCUの命令で直接アクセスできるように構成でき、比較的簡単な回路で実現できる。

【0226】以上のように、第7の実施の形態によれば、比較的の小規模な回路で、演算終了割り込み機能を持った、外部インタフェース付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内蔵のMCUを構成でき、しかもLSI化が可能となる。

【0227】（第8の実施形態）本コプロセッサ内での剰余演算は、演算開始から終了までダイナミックに実行

されるため、外部装置と接続した際のシステム全体の演算実行中の消費電流を抑えるために、演算実行時の外部装置の動作を一時停止（以下、スリープと呼ぶ。）状態にすると都合がよい。

【0228】本提案は、それを実現するために、専用の外部装置スリープ制御回路とクロック制御回路を設けたものである。

【0229】図20は、第8の実施形態を示す図であり、第7の実施形態に、外部装置スリープ制御回路とクロック制御回路を付加したものである。

【0230】図20中で、SlpConはMCUスリープ制御回路であり、MCUからのスリープ設定信号Slpsetを受けてスリープ信号Slpを発生させ、その信号をクロック制御回路CLKConに送る。

【0231】CLKConは、通常は外部より入力されたシステムクロックCLK信号をもとに、MCUに対してMCU原振クロックMCUCLKを供給しているが、いったんSlp信号を受け取るとMCUへのMCUCLKの供給をやめるように働く。

【0232】演算終了時には、演算終了のタイミング信号CoendがSlpConに入力されてSlp信号のCLKConへの供給がなくなり、CLKConからMCUに対してMCUCLKの供給を再開する。

【0233】なお、このスリープ機能は一般的には外部装置のある端子への信号入力等でスリープ解除することが多く、この場合もそれを併用するようにSlpConを構成しても良い。

【0234】SlpConは、演算コントロール回路CopConと同様にMCUのペリフェラル回路として、ローカルメモリエリア等に割り付けてMCUの命令で直接アクセスできるように構成でき、比較的簡単な回路で実現できる。

【0235】以上のように、第8の実施形態によれば、比較的の小規模な回路で、外部装置スリープ機能を持った、外部インターフェイス付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内蔵のMCUを構成でき、システム全体の低消費電力化が実現して、しかもLSI化が可能となる。

【0236】（第9の実施形態）本コプロセッサの用途は、大規模で複雑な剰余演算を行う暗号アルゴリズム等の、処理時間の高速性を問われる場合に使用することができる。従って、システムに供給される入力原振クロックの周波数がいかなる場合であっても、演算時間は短いことが要求される。

【0237】そこで、通倍速クロック制御回路を内蔵して演算速度の向上機能を付加する。

【0238】図21は、第9の実施の形態を示す図であり、第8の実施の形態のクロック制御回路をモディファイして通倍速クロック制御回路を構成した場合の図である。なお、第6、第7の実施の形態に通倍速クロック制

御回路を別個に付加した場合でも構わない。

【0239】図21中、CLKCon2は、通倍速クロック制御回路を内蔵したクロック制御回路であり、MCUから通倍速設定用の信号としてCkwsetが入力されると通倍速機能が働く。

【0240】通倍速機能は、外部から供給される原振クロックCLKに対して動作し、作成した通倍速クロックがMCUの原振クロックMCUCLK、あるいはコプロセッサの原振クロックCOPCLK、あるいはその両方に出力させるように回路を構成しておく。そうすることで、ユーザーにとっては、消費電流との兼ね合いから、システム上でバリエーションに富んだ選択ができるようになる。

【0241】通倍速機能の解除も、設定の場合と同様に解除信号としてのCkwset信号を入力することによって、実現する。

【0242】通倍速クロック制御回路は、演算コントロールCopConと同様にMCUのペリフェラル回路としてローカルメモリエリア等に割り付けてMCUの命令で直接アクセスできるように構成でき、比較的簡単な回路で実現できる。

【0243】以上のように、第9の実施の形態によれば、比較的の小規模な回路で、システムの通倍速機能を持った、外部インタフェース付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内臓のMCUを構成でき、システム全体の演算処理速度が向上して、しかもLSI化が可能となる。

【0244】第1から第5の実施の形態で示した高速乗加算器Mul/Addは、乗算器と加算器に分離した構成としても良い。

【0245】第1から第5の実施の形態で示した高速乗加算器Mul/Add、高速加算器Addは、一体化されたものであっても良い。

【0246】第1から第5の実施の形態で示した高速乗加算器Mul/Add、高速加算器Addは、市販のICで構成しても良い。

【0247】第6から第9の実施の形態で示したコプロセッサは、第1から第5の実施の形態で示したコプロセッサをもとにした記述をしているが、そのコプロセッサと同等の機能を持つ他のコプロセッサで構成しても良い。

【0248】第1から第5の実施の形態で示した多ビット長乗算アルゴリズムは、一般的なものでは説明しているが、説明の中で使用しているハードウェア構成で利用できる他の乗算アルゴリズム（例えば、BOOTH等）をタイミング/コントロール回路で作り出して実現しても良い。

【0249】第2から第5の実施の形態の説明では、“0”検出回路ZeroCの使用例を演算値メモリの削減を主な目的として記述しているが、ZeroCは例えば

単位乗算の乗算値あるいは被乗算値の内容が0であった場合に、演算動作をオミットし演算結果の途中値を無条件に0として次の演算に進むようにシーケンスを制御して、演算時間を短縮する目的にも使用できる。

【0250】第5の実施の形態では、被乗算値格納レジスタの個数を増やす記述をしているが、被乗算値格納レジスタのビット長を大きくしてサイズを増やす場合も可能である。

【0251】

10 【発明の効果】以上詳細に説明したように、本発明の代表的なものによれば、べき乗剰余演算の解を求めるに際し、予め用意されたモード信号を供給するのみで、各種演算を実行することができる。

【図面の簡単な説明】

【図1】本発明におけるフローチャートを示す図である。

【図2】暗号文の処理の具体例を説明する図である。

【図3】べき乗剰余演算の概略フローである。

20 【図4】本発明のハードウェアの概略を示すブロック図である。

【図5】本発明の第1の実施の形態を示す図である。

【図6】単位乗加算を説明する図である。

【図7】本発明の第1の実施形態の動作を説明するハードウェアイメージである。

【図8】本発明の第2の実施の形態を示す図である。

【図9】本発明の第3の実施の形態を示す図である。

【図10】本発明の第4の実施の形態を示す図である。

【図11】本発明の第4の実施形態の動作を説明するハードウェアイメージである。

30 【図12】本発明の第4の実施形態の動作を説明する演算例である。

【図13】本発明の第5の実施の形態を示す図である。

【図14】本発明の第5の実施形態の動作を説明するハードウェアイメージである。

【図15】本発明の第5の実施形態の動作を説明するハードウェアイメージである。

【図16】本発明の第5の実施形態の動作を説明するハードウェアイメージである。

40 【図17】本発明の第5の実施形態の動作を説明する演算例である。

【図18】本発明の第6の実施の形態を示す図である。

【図19】本発明の第7の実施の形態を示す図である。

【図20】本発明の第8の実施の形態を示す図である。

【図21】本発明の第9の実施の形態を示す図である。

【符号の説明】

T/C・・・タイミング/コントロール回路

Smem、N'mem、Nmem、Mmem、A'mem、

Wlmem、Whmem・・・演算値メモリ

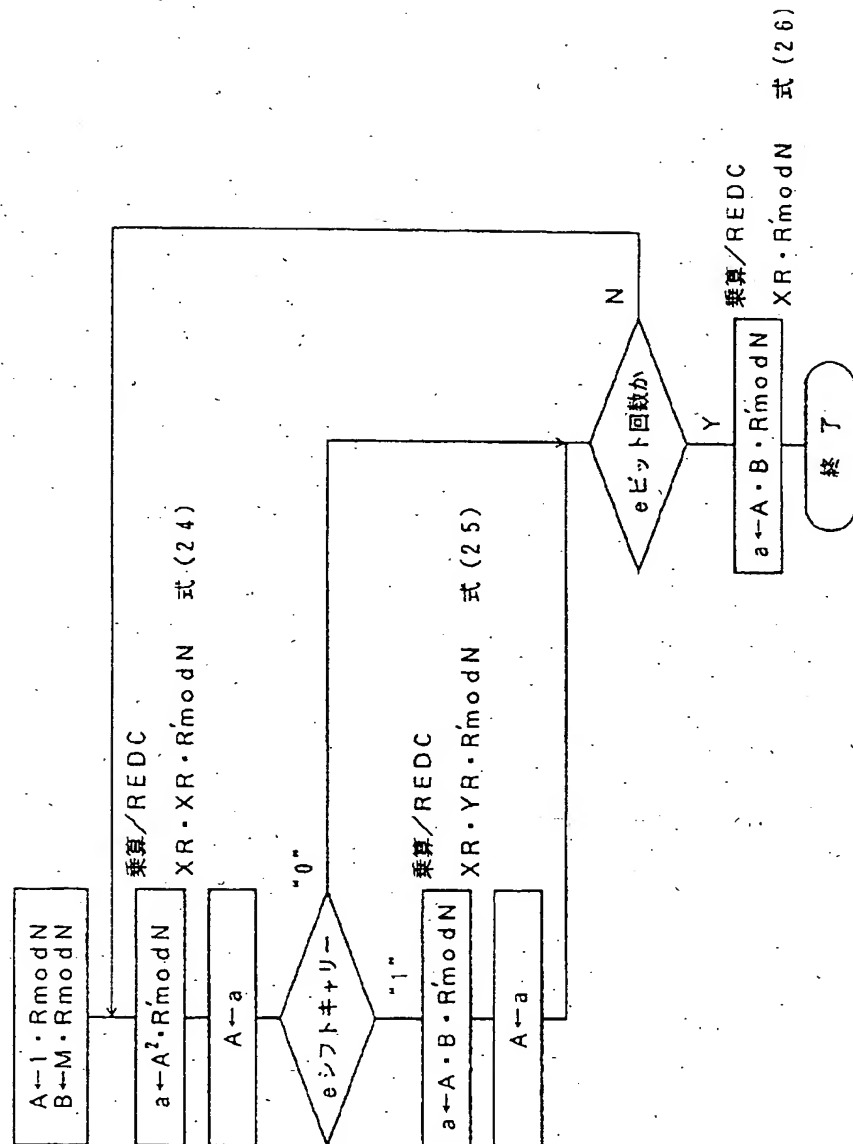
Mul/Add・・・高速乗加算器

50 Add・・・高速加算器

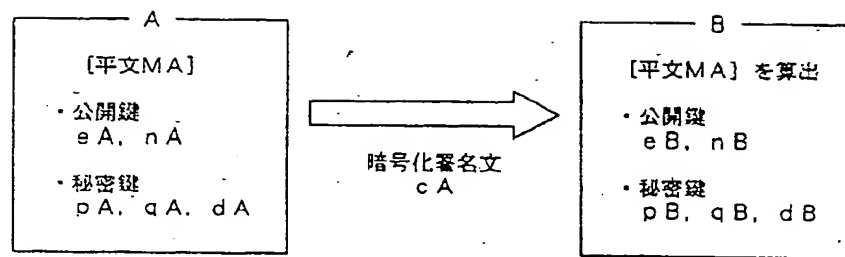
Xi-reg . . . 乗算値格納レジスタ
 Yi-reg . . . 被乗算値格納レジスタ
 Ai-reg . . . 被加算値格納レジスタ

RH . . . 高速加算器Addが出力する値の上位の桁を
 格納するレジスタ

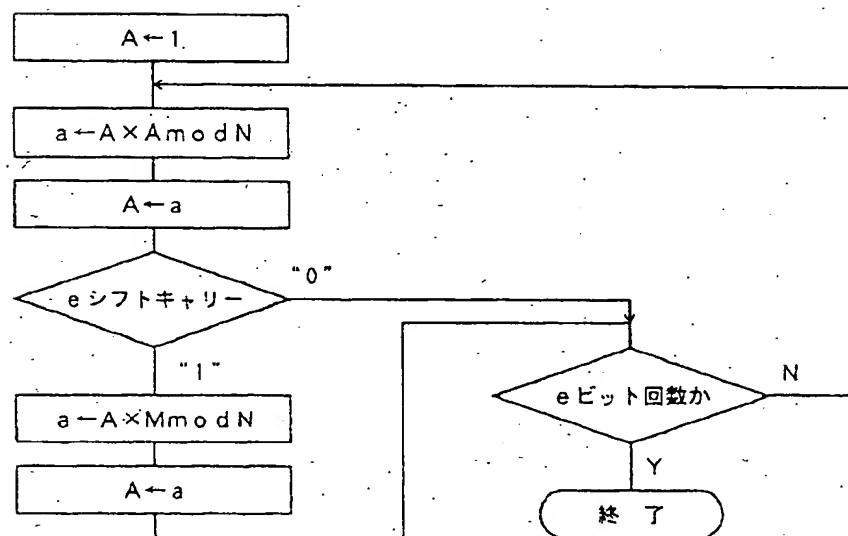
【図1】



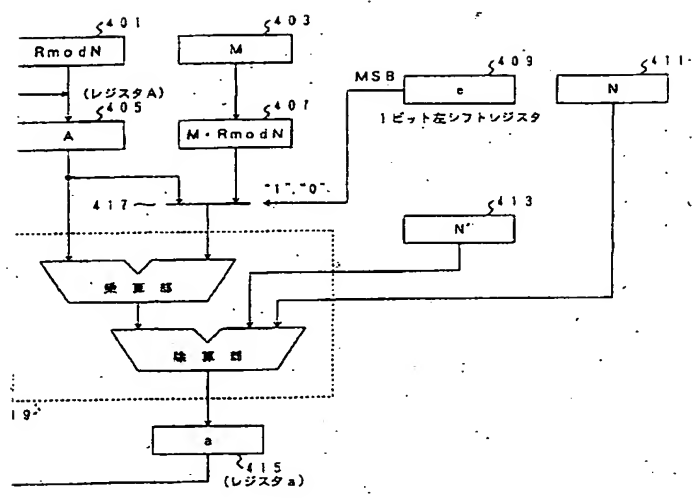
【図2】



【図3】

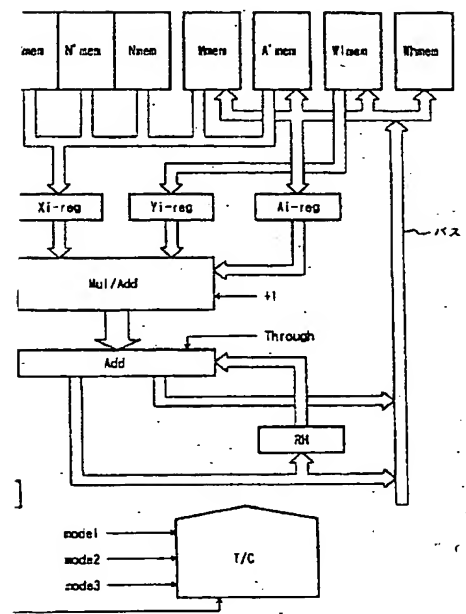
べき乗剰余演算 ($M \cdot \text{mod } N$) 概略フロー

【図4】



【図5】

【図6】



乗算入力処理ビット長 (4 ビット)

$$A \times A = \overbrace{A'3 A'2 A'1 A'0} \times \underbrace{W13 W12 W11 W10}$$

演算ビット長 (16ビット)

$$\begin{array}{r} A'3 A'2 A'1 A'0 \leftarrow A' \text{ mem の値} \\ \times) \quad W13 W12 W11 W10 \leftarrow W \text{ mem の値} \\ \hline A'0 \times W10 \leftarrow \text{単位乗加算} \\ A'1 \times W10 \\ A'2 \times W10 \\ A'3 \times W10 \\ A'0 \times W11 \\ A'1 \times W11 \end{array}$$

第1の実施の形態

【図7】

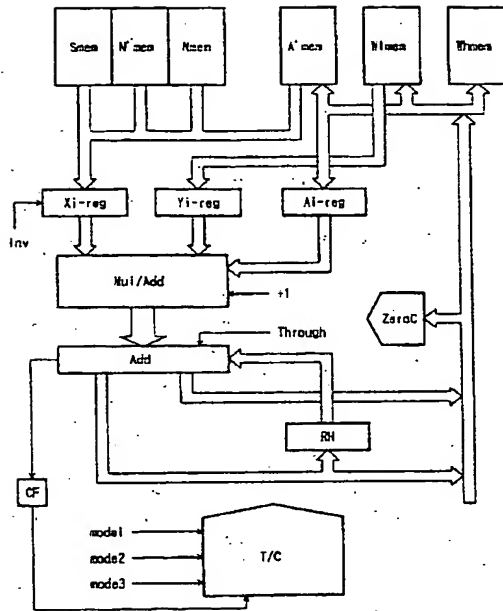
$$A'3 A'2 A'1 A'0 \times W13 W12 W11 W10 = Wh3 Wh2 Wh1 Wh0 W13 W12 W11 W10$$

- | | |
|--|--|
| ① $A'0 \times W10 + 0 = RH \underline{W10}$ | ⑨ $A'0 \times W12 + Wh0 = RH \underline{W12}$ |
| ② $A'1 \times W10 + 0 + RH = RH \underline{Wh0}$ | ⑩ $A'1 \times W12 + Wh1 + RH = RH \underline{Wh0}$ |
| ③ $A'2 \times W10 + 0 + RH = RH \underline{Wh1}$ | ⑪ $A'2 \times W12 + Wh2 + RH = RH \underline{Wh1}$ |
| ④ $A'3 \times W10 + 0 + RH = RH \underline{Wh2}$ | ⑫ $A'3 \times W12 + Wh3 + RH = RH \underline{Wh2}$ |
| ⑤ $A'0 \times W11 + Wh0 = RH \underline{W11}$ | ⑬ $A'0 \times W13 + Wh0 = RH \underline{W13}$ |
| ⑥ $A'1 \times W11 + Wh1 + RH = RH \underline{Wh0}$ | ⑭ $A'1 \times W13 + Wh1 + RH = RH \underline{Wh0}$ |
| ⑦ $A'2 \times W11 + Wh2 + RH = RH \underline{Wh1}$ | ⑮ $A'2 \times W13 + Wh2 + RH = RH \underline{Wh1}$ |
| ⑧ $A'3 \times W11 + Wh3 + RH = RH \underline{Wh2}$ | ⑯ $A'3 \times W13 + Wh3 + RH = RH \underline{Wh2}$ |

[Wmem-WImem *]

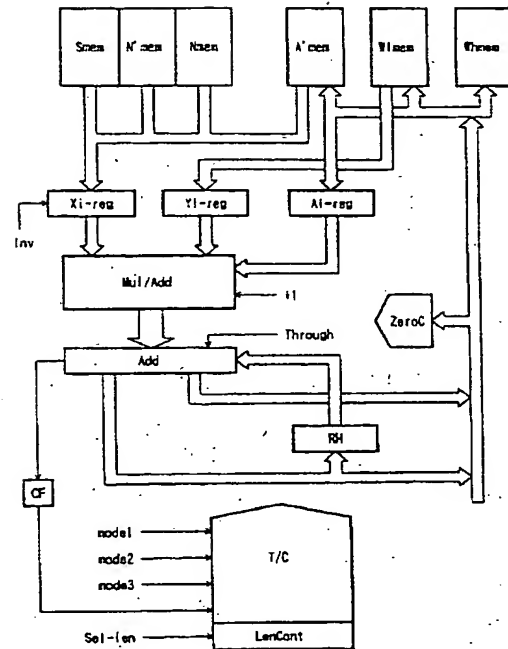
例えば 8ビット × 8ビット の計算 $6AH \times F5H = 6572H$ の場合の計算結果 6572H について、65H を Wmem に、72H を WImem にそれぞれ格納することを示す。

【図8】



第2の実施の形態

【図9】



第3の実施の形態

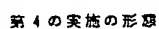
【図11】

$$S3 \ S2 \ S1 \ S0 \times \ W13 \ W12 \ W11 \ W10 + \ Wb3 \ Wb2 \ Wb1 \ Wb0 = \ Wb3 \ Wb2 \ Wb1 \ Wb0 \ W13 \ W12 \ W11 \ W10$$

- ① $S0 \times W10 + Wb0 = RH \ W10$
- ② $S1 \times W10 + Wb1 + RH = RH \ Wb0$
- ③ $S2 \times W10 + Wb2 + RH = RH \ Wb1$
- ④ $S3 \times W10 + Wb3 + RH = RH \ Wb2$
- ⑤ $S0 \times W11 + Wb0 = RH \ W11$
- ⑥ $S1 \times W11 + Wb1 + RH = RH \ Wb0$
- ⑦ $S2 \times W11 + Wb2 + RH = RH \ Wb1$
- ⑧ $S3 \times W11 + Wb3 + RH = RH \ Wb2$

- ⑨ $S0 \times W12 + Wb0 = RH \ W12$
- ⑩ $S1 \times W12 + Wb1 + RH = RH \ Wb0$
- ⑪ $S2 \times W12 + Wb2 + RH = RH \ Wb1$
- ⑫ $S3 \times W12 + Wb3 + RH = RH \ Wb2$
- ⑬ $S0 \times W13 + Wb0 = RH \ W13$
- ⑭ $S1 \times W13 + Wb1 + RH = RH \ Wb0$
- ⑮ $S2 \times W13 + Wb2 + RH = RH \ Wb1$
- ⑯ $S3 \times W13 + Wb3 + RH = RH \ Wb2$

【图 1 2】

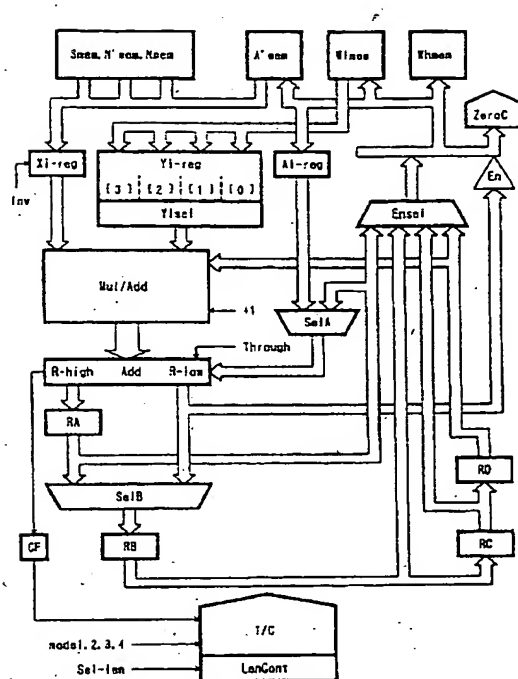


[illegible]

【图 17】

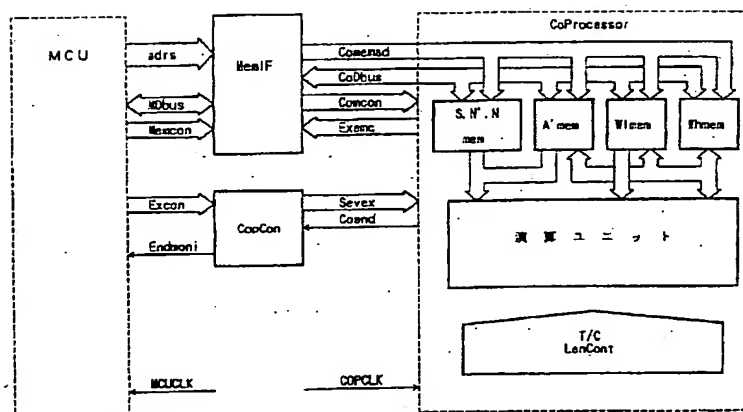
[illegible]

【图 13】



第 5 の実施の形態

【图 18】



第 8 の実施の形態

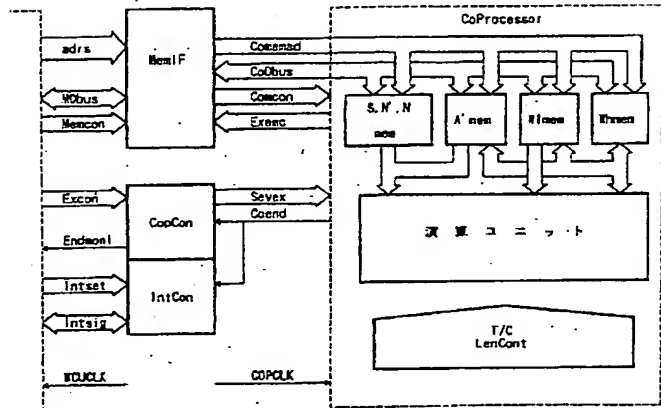
【図14】

$$A3 A2 A1 A0 \cdot B0 + C0$$

$$W13 W12 W11 W10 \times S0 + Wh0 = RA RB RC RD W10$$

設定]
A ← RB = RC = RD = "0" , Xi-reg ← S0 , Ai-reg ← Wh0
i-reg[0] ← W10 , Yi-reg[1] ← W11 , Yi-reg[2] ← W12 , Yi-reg[3] ← W13
[Step1]
Xi-reg × Yi-reg[0] + RD + Ai-reg = (R-high)(R-low)
RD ← RC , RC ← RB , RB ← RA
RA ← (R-high) , W10 ← (R-low)
[Step2]
Xi-reg × Yi-reg[1] + RD + RA = (R-high)(R-low)
RD ← RC , RC ← RB
RA ← (R-high) , RB ← (R-low)
[Step3]
Xi-reg × Yi-reg[2] + RD + RA = (R-high)(R-low)
RD ← RC , RC ← RB
RA ← (R-high) , RB ← (R-low)
[Step4]
Xi-reg × Yi-reg[3] + RD + RA = (R-high)(R-low)
RD ← RC , RC ← RB
RA ← (R-high) , RB ← (R-low)

【図19】



第7の実施の形態

【図15】

$$A3 A2 A1 A0 + B3 B2 B1 B0 + C3 C2 C1 C0$$

$$W13 W12 W11 W10 \times S3 S2 S1 S0 + Wh3 Wh2 Wh1 Wh0 = RA RB RC RD W13 W12 W11 W10$$

[初期設定]

$RA = RB = RC = RD = "0"$, $Xi-reg \leftarrow S0$, $Ai-reg \leftarrow Wh0$

$Yi-reg[0] \leftarrow W10$, $Yi-reg[1] \leftarrow W11$, $Yi-reg[2] \leftarrow W12$, $Yi-reg[3] \leftarrow W13$

<Time1>

[Step1]

$Xi-reg \times Yi-reg[0] + RD + Ai-reg = (R-high)(R-low)$

$RD \leftarrow RC$, $RC \leftarrow RB$, $RB \leftarrow RA$, $RA \leftarrow (R-high)$, $W10 \leftarrow (R-low)$

[Step2]

$Xi-reg \times Yi-reg[1] + RD + RA = (R-high)(R-low)$

$RD \leftarrow RC$, $RC \leftarrow RB$, $RA \leftarrow (R-high)$, $RB \leftarrow (R-low)$

[Step3]

$Xi-reg \times Yi-reg[2] + RD + RA = (R-high)(R-low)$

$RD \leftarrow RC$, $RC \leftarrow RB$, $RA \leftarrow (R-high)$, $RB \leftarrow (R-low)$

[Step4]

$Xi-reg \times Yi-reg[3] + RD + RA = (R-high)(R-low)$

$RD \leftarrow RC$, $RC \leftarrow RB$, $RA \leftarrow (R-high)$, $RB \leftarrow (R-low)$

* Step2 から Step4 までの期間に、各メモリのアドレスの変更、プリチャージ、
Xi-reg へのデータ格納、Ai-reg へのデータ格納をそれぞれ行う。

$Xi-reg \leftarrow S1$, $Ai-reg \leftarrow Wh1$

【図 16】

<Time2>

[Step1]

W11 ← (R-low) の操作以外は Time1 の場合と同様。

[Step2] ~ [Step4]

Time1 の場合とまったく同様。

$$X_{i-reg} \leftarrow S2, \quad A_{i-reg} \leftarrow Wh2$$

<Time3>

[Step1]

W12 ← (R-low) の操作以外は Time1 の場合と同様。

[Step2] ~ [Step4]

Time1 の場合とまったく同様。

$$Xi-reg \leftarrow S3, \quad Ai-reg \leftarrow Wh3$$

<Time4>

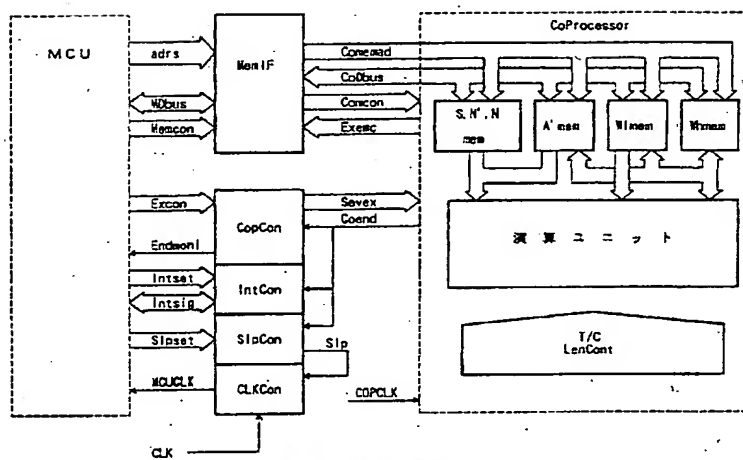
[Step1]

W13 ← (R-low) の操作以外は Time1 の場合と同様。

[Step2] ~ [Step4]

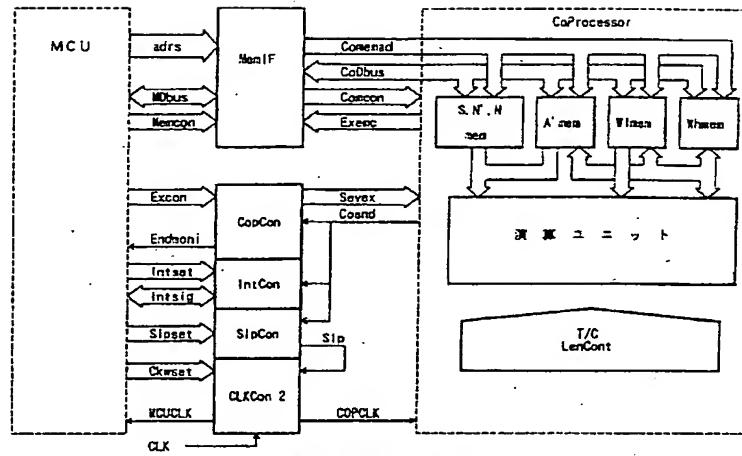
Time1 の場合とまったく同様。

【圖20】



第 8 の実施の形態

【図21】



第9の実施の形態

フロントページの続き

(72)発明者 川▲崎▼ 清人
宮崎県児湯郡高鍋町大字南高鍋569番地3
合資会社川▲崎▼電機内